1988-12

A CAD tool for current-mode multiple-valued CMOS circuits

Lee, Hoon S.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A CAD TOOL FOR CURRENT-MODE MULTIPLE-VALUED CMOS CIRCUITS

by

Hoon S. Lee

December 1988

Thesis Advisor: Jon T. Butler

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704 0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
A CAD TOOL FOR CURRENT-MODE MULTIPLE-VALUED CMOS CIRCUITS

12 PERSONAL AUTHOR(S)
Lee, Hoon S.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 December	123

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	CAD (computer aided design), logic design, CMOS logic circuits, logic minimization, sum-of-products expressions, programmable logic array (PLA)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The contribution of this thesis is the development of a CAD (computer aided design) tool for current mode multiple-valued logic (MVL) CMOS circuits. It is only the second known MVL CAD tool and the first CAD tool for MVL CMOS.

The tool accepts a specification of the function to be realized by the user, produces a minimal or near-minimal realization (if such a realization is possible), and produces a layout of a programmable logic array (PLA) integrated circuit that realizes the given function. The layout is in MAGIC format, suitable for submission to a chip manufacturer. The CAD tool also allows the user to simulate the realized function so that he/she can verify correctness of design.

The CAD tool is designed also to be an analysis tool for heuristic minimization algorithms. As part of this thesis, a random function

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Jon T. Butler	22b TELEPHONE (Include Area Code) 408-646-3299	22c OFFICE SYMBOL Code 62Bu

DD Form 1473, JUN 86 *Previous editions are obsolete*
S/N 0102-LF-014-6603

18. Cont.

multiple-valued logic

19. Cont.

generator and statistics gathering package were developed. In the present tool, two heuristics are provided and the user can choose one or both. In the latter case, the better realization is output to the user. The CAD tool is designed to be flexible, so that future improvements can be made in the heuristic algorithms, as well as the layout generator. Thus, the tool can be used to accommodate new technologies, for example, a voltage mode CMOS PLA rather than the current mode CMOS currently implemented.

Approved for public release; distribution is unlimited

A CAD Tool for Current-Mode Multiple-Valued CMOS Circuits

by

Hoon S. Lee
Lieutenant, Republic of Korea Navy
B.S., Naval Academy, Chin-hae, Korea, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

December 1988

ABSTRACT

The contribution of this thesis is the development of a CAD (computer aided design) tool for current mode multiple-valued logic (MVL) CMOS circuits. It is only the second known MVL CAD tool and the first CAD tool for MVL CMOS.

The tool accepts a specification of the function to be realized by the user, produces a minimal or near-minimal realization (if such a realization is possible), and produces a layout of a programmable logic array (PLA) integrated circuit that realizes the given function. The layout is in MAGIC format, suitable for submission to a chip manufacturer. The CAD tool also allows the user to simulate the realized function so that he/she can verify correctness of design.

The CAD tool is designed also to be an analysis tool for heuristic minimization algorithms. As part of this thesis, a random function generator and statistics gathering package were developed. In the present tool, two heuristics are provided and the user can choose one or both. In the latter case, the better realization is output to the user. The CAD tool is designed to be flexible, so that future improvements can be made in the heuristic algorithms, as well as the layout generator. Thus, the tool can be used to accommodate new technologies, for example, a voltage mode CMOS PLA rather than the current mode CMOS currently implemented.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my sincere appreciation to Dr. J. T. Butler, my thesis advisor, and to Dr. Chyan Yang, my second reader, both of whom are from the Department of Electrical and Computer Engineering of the Naval Postgraduate School, and also to LCDR J. M. Yurchak, my coadvisor from the Department of Computer Science of the Naval Postgraduate School, for their great academic guidance as well as encouragement in preparation of this thesis.

I also wish to express my great thanks to my wife, Hye-sook, and my son, Chu-son, for their support and patience away from home during these two years in the United States.

I. INTRODUCTION

A computer-aided design (CAD) tool has been created at the Naval Postgraduate School to design multiple-valued logic programmable logic arrays (MVL-PLA). The CAD tool accepts a function specification and produces a circuit layout. Small MVL-PLA's can be designed by hand. However, for most practical applications, hand design is too time consuming and prone to errors.

The CAD tool performs the following:

- Accepts an input function specification, specifically an algebraic expression.
- Minimizes the expression producing a realization that occupies the least amount of space.
- Produces a layout which can be used to generate the MVL-PLA chip.

The CAD tool is designed to accommodate future improvements in:

- The minimization program and
- MVL-PLA layout.

It is expected that as knowledge is gained about these aspects of MVL-PLA design, there will be new subroutines that will substitute for these parts of the tool. The whole program has been written so that such changes can be easily made.

This CAD tool, which applies to current-mode CMOS technology, is only the second known tool for multiple-valued logic circuit design. A CAD tool exists for the design of multiple-valued CCD PLA design [Ref. 1].

A. BACKGROUND

The promise of current-mode CMOS circuits for compact realizations has generated much interest. Japanese researchers have developed a binary multiplier integrated circuit using bi-directional current-mode multiple-valued CMOS [Ref. 2]. The circuit has speed almost equal to the fastest binary multiplier [Ref. 3] but requires half the chip area and dissipates half the power. Researchers in Holland, France and the United States have also fabricated current mode multiple-valued CMOS circuits.

Since this technology is new, no design tool has been developed for it. Design tools are absolutely needed to produce practical circuits since modern VLSI circuits are so complicated. It is the lack of such tools that has motivated the research described in this thesis.

II. PROGRAM DEVELOPMENT

The package developed in this thesis is a comprehensive program written in the "C" language to run on the VAX-11/785 computer at the Naval Postgraduate School. As "C" is highly structured, the program is easy to maintain and easy to develop into a large program package. Also, use of "C" assures compatability with presently available binary VLSI design tools such as the Berkeley CAD tools [Ref. 4].

A. APPROACH TO THE PROBLEM

A top-down programming technique was used. First, the tasks were defined which the program needed to perform. Then, each task was developed through a series of steps to identify very specific routines required to perform each task.

The major tasks are as follows:

- Data Entry
- Logic Minimization
- Logic Simulation
- Layout Generation

1. Data Entry

- Allows the user to enter input expression from an input data file.
- Partitions input expression into identifiable parts, called tokens.
- Examines each token for errors.
- Flags errors when they occur.

2. Logic Minimization

- Three minimization algorithms are used.
  - * Pomper and Armstrong [Ref. 5].
  - * Dueck and Miller [Ref. 6].
  - * Gold [Ref. 7].
- All of them are heuristic and based on the direct cover approach.
- Minimization produces a minimal or near minimal solution, thus reducing the chip area needed.

This routine will be discussed in more detail in the following section.

3. Logic Simulation

- Verifies the minimal solution from the Logic Minimization routine by applying test vectors given by the user to it.

4. Layout Generation

- Uses a minimal solution from the Logic Minimization routine to produce a layout which will be used for MVL-PLA chips.

B. ORGANIZATION OF THE PROGRAM

The basic organization of the program is shown in Figure 2.1. The solid lines show flow of control. The user supplies input expressions via the data entry routine. These are applied to the logic minimization routine for obtaining the minimized expression. The minimized expression is then passed to the logic simulation routine for verification. Then it is used for layout generation for MVL-PLA chips.

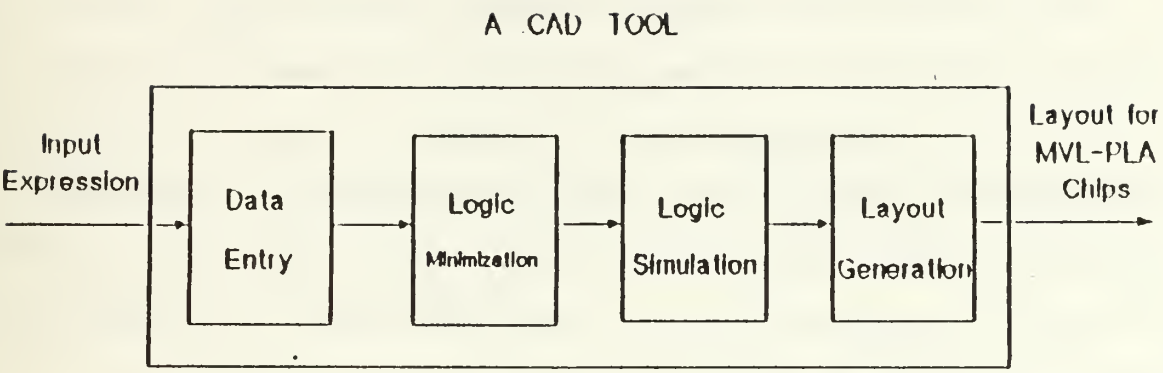

Figure 2.1: Program Organization

PLA design using this program resembles program development under an operating system. An editor is used to develop the source file (input expression). This is compiled by the data entry routine. If it is free of syntax errors, it is assembled (minimized) and run (a layout is produced).

III. PROGRAM DESCRIPTION

The CAD tool is best described by considering a specific example. The user starts with a logic diagram (or Karnaugh map) of the input expression to be realized. An example of the logic diagram is shown in Figure 3.1. It shows a 4-valued 2 variable expression. Without a CAD program, an inexperienced user might spend half an hour to find a minimal solution of the expression. What is worse, when the expression is complicated because of a large radix and/or a large number of variables, it may be impossible not only to find the minimal solution, but also to visualize it.

Since the CAD program is designed to find a minimal solution of the expression, the user is not required to minimize the expression. All the user has to do is specify the expression correctly. Figure 3.2 shows one way the function of Figure 3.1 can be expressed for the CAD tool. Here each circle represents a product term. Equation 3.1 shows an algebraic equivalent of the function of Figure 3.2.

$$
\begin{aligned}
&4:2: \qquad (3.1)\\
&+2 * X1(1,2) * X2(0,0)\\
&+3 * X1(3,3) * X2(1,1)\\
&+2 * X1(0,0) * X2(1,2)\\
&+1 * X1(0,0) * X2(2,3)\\
&+2 * X1(1,1) * X2(1,1)\\
&+2 * X1(1,1) * X2(2,3)
\end{aligned}
$$

$$+1 * X1(1,1) * X2(2,2)$$

$$+2 * X1(2,3) * X2(1,1)$$

$$+1 * X1(2,2) * X2(1,2)$$

$$+1 * X1(3,3) * X2(2,2)$$

$$+3 * X1(2,2) * X2(3,3)$$

Equation 3.1 is represented according to the standard format of the expression shown in Table 3.1. Any multiple-valued logic function $f(X1, X2, X3, \cdots, Xn)$ can be represented by the following format:

$$\begin{aligned} f = \quad & \text{radix : number of variables :} \\ & +c_1 * X1(a1, b1) * X2(a2, b2) * \cdots * Xn(an, bn) \\ & +c_2 * X1(c1, d1) * X2(c2, d2) * \cdots * Xn(cn, dn) \\ & \vdots \\ & +c_n * X1(f1, g1) * X2(f2, g2) * \cdots * Xn(fn, gn) \end{aligned} \tag{3.2}$$

where $c_1, c_2, \cdots and\ c_n$ denote a logic value between '1' and '(radix -1)', the symbol '*' denotes the minimum operator, $Xn(an, bn)$ denotes a literal function of the input variable 'Xn' with lower bound 'an' and upper bound 'bn'. Specifically, $Xn(an, bn)$ is (radix -1) if $an \leq Xn \leq bn$, and 0 otherwise. The symbol '+' represents the truncated arithmatic sum (truncated to (radix -1) if the actual sum exceeds (radix - 1)).

According to Equation 3.1, the expression has radix 4 and 2 variables and a list of terms (11 terms in this expression). A term is composed of a coefficient and a list of variables (2 variables for each term in this case). The expression shown in Equation 3.1 is entered into the CAD system via the data file.

X2 \ X1	0	1	2	3
0		2	2	3
1	2	2	3	2
2	3	3	1	1
3	1	2	3	

Figure 3.1: An Example of Logic Diagram for 4-Valued 2 Variable Expression

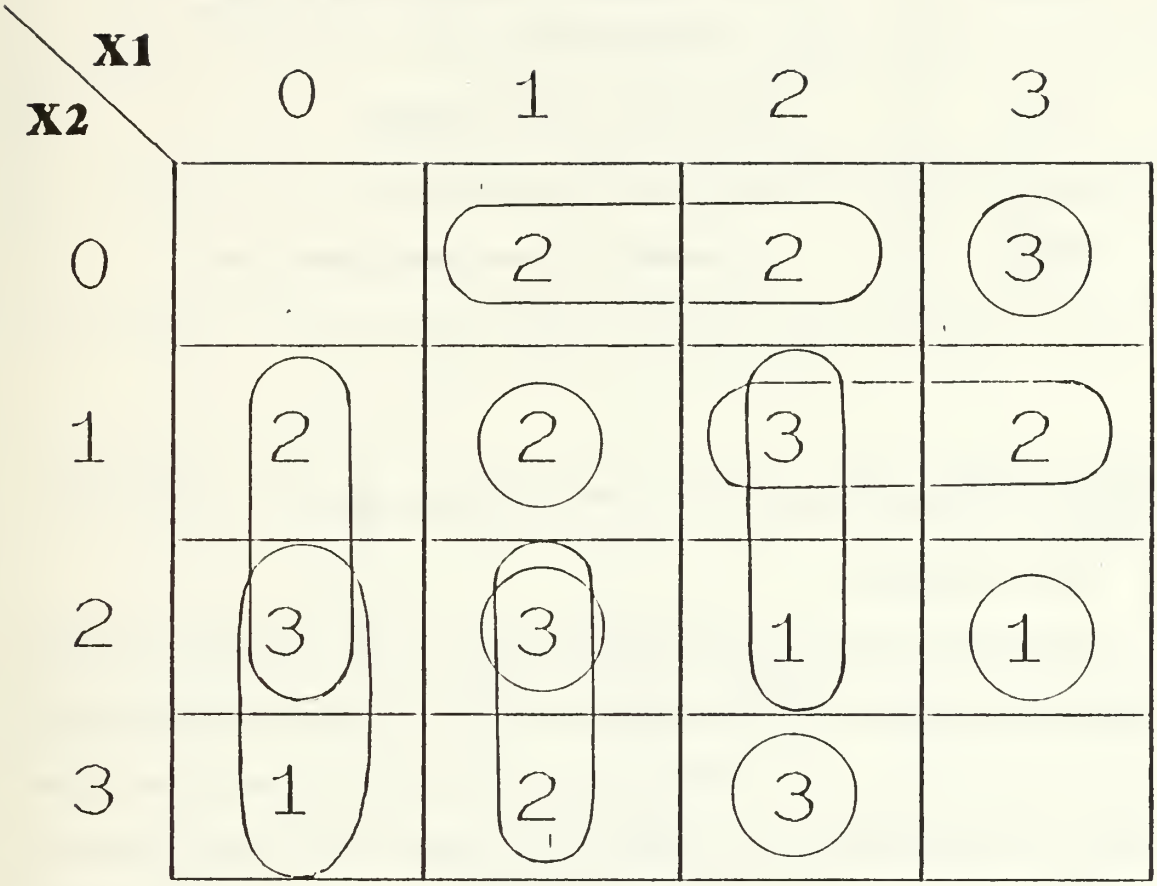

Figure 3.2: An Example of Logic Diagram for An Input Expression Specified by the User

Expression	Radix : Number of Variables : A List of Terms
Term	Coefficient * A List of Variables
Variable	X (Variable ID) (Lower Bound, Upper Bound)

TABLE 3.1: Standard Format of an Input Expression for a CAD Tool

A. DATA ENTRY

Figure 3.3 shows the flowchart of the Data Entry routine. The input expression from the input data file is applied to this routine first. It examines all the expressions in the data file. There is a special function, called the parser/scanner in this routine. It partitions the expression into identifiable parts, called tokens. Then, it passes each token to the next three consecutive error checking subroutines – the syntax error checker, the grammar error checker, and the range error checker.

If an error is found, the corresponding error message is presented to the user and the program execution stops. The user needs to go back to the input data file and correct it. If all the expressions are correct, all are stored in the allocated memory space. Figure 3.4 shows the data structure of the input expression. It is developed to accommodate the input expression.

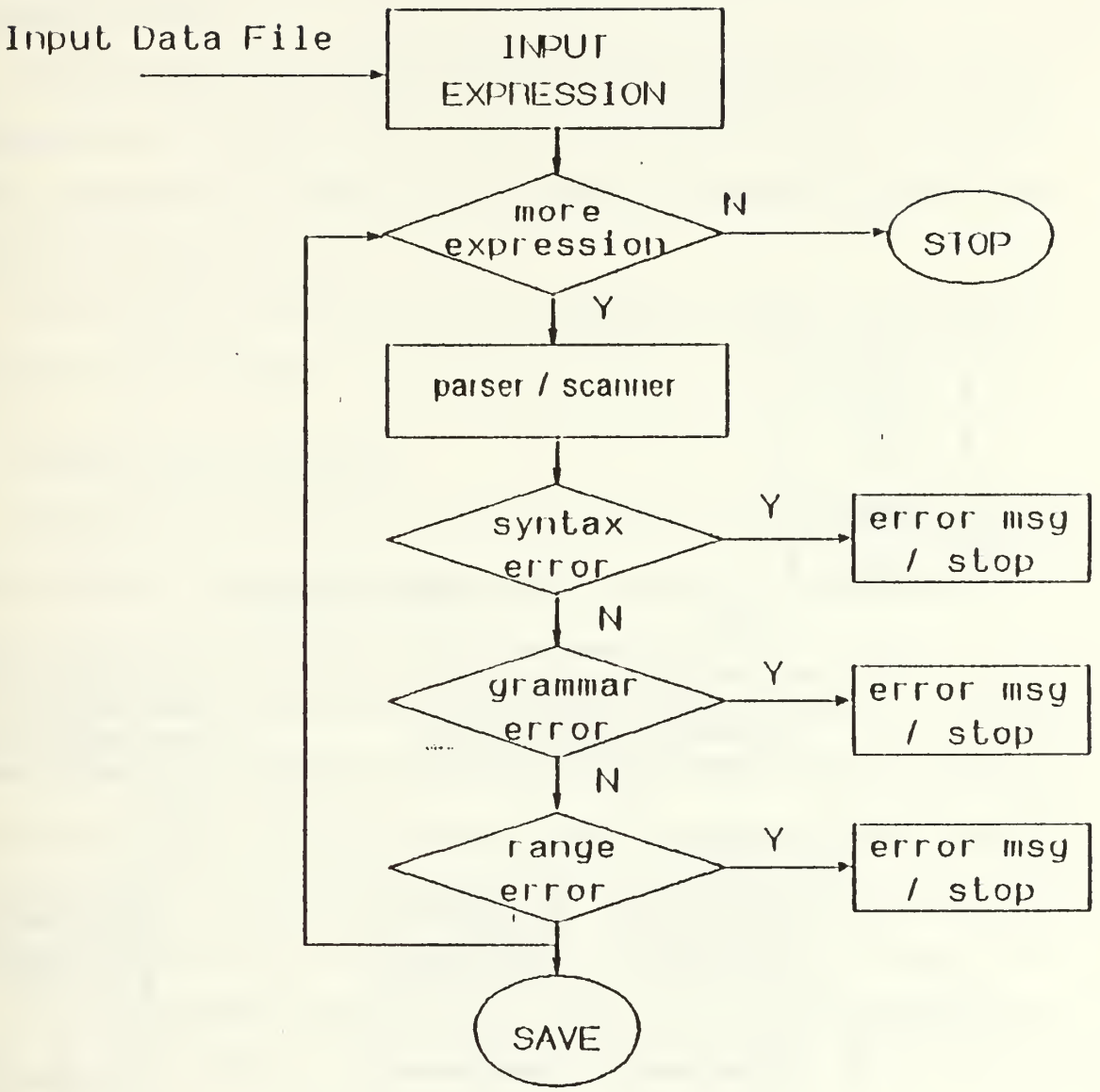

Figure 3.3: Flow Chart of the Data Entry Routine

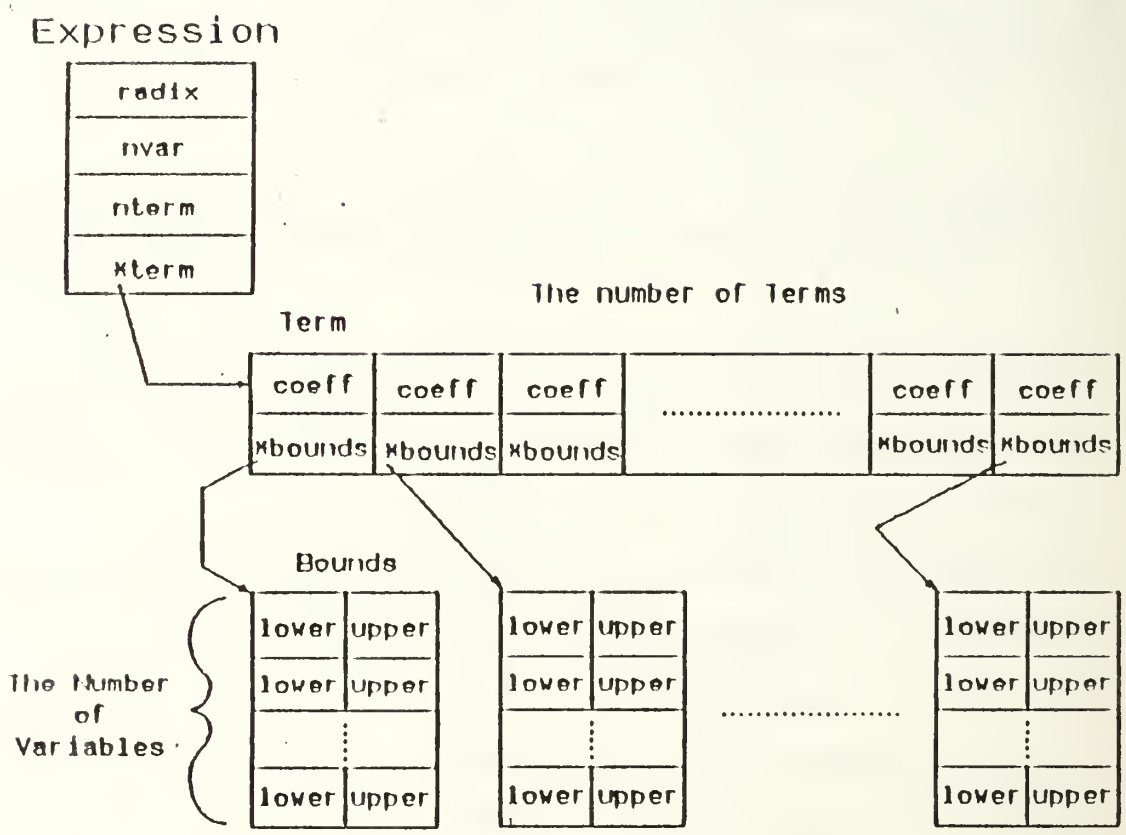

Figure 3.4: Data Structure of the Input Expression

According to the Figure 3.4, three data structures form a complete input expression. They are the Expression, Term and Bounds data structures. The radix, number of variables and number of terms in the input expression are saved in the Expression structure.

The size of the array for the term lists is determined by the number of terms of the expression. Also, the size of the bounds array is equal to the number of variables used in the expression. Therefore, all the components of the input expression are saved into this data structure.

Once this data structure for the input expression is formed, it will be used throughout the CAD program.

B. LOGIC MINIMIZATION

The data structure of the input expression is formed in the Data Entry routine. This is then used by Logic Minimization routine to minimize the expression. Figure 3.5 shows the flowchart of the Logic Minimization routine.

First, a working copy (f) of the original input expression is made. If f is fully covered (made up of only zeroes and don't care terms (radix)), the program stops. The minimization process is accomplished. Otherwise, the program finds the most isolated minterm first, then picks the best implicant (or product term) from among the possible implicants which cover the chosen most isolated minterm.

The best implicant chosen is saved as a part of the minimal solution of the input expression. Then it is taken out from the working copy (f). This process is repeated until the resulting expression has no more minterms.

As mentioned earlier, three heuristic algorithms are used in this thesis. As shown in Reference 7, neither that of the Dueck and Miller [Ref. 6] nor that of and Pomper and Armstrong [Ref. 5] is consistently better than the other. That is, there

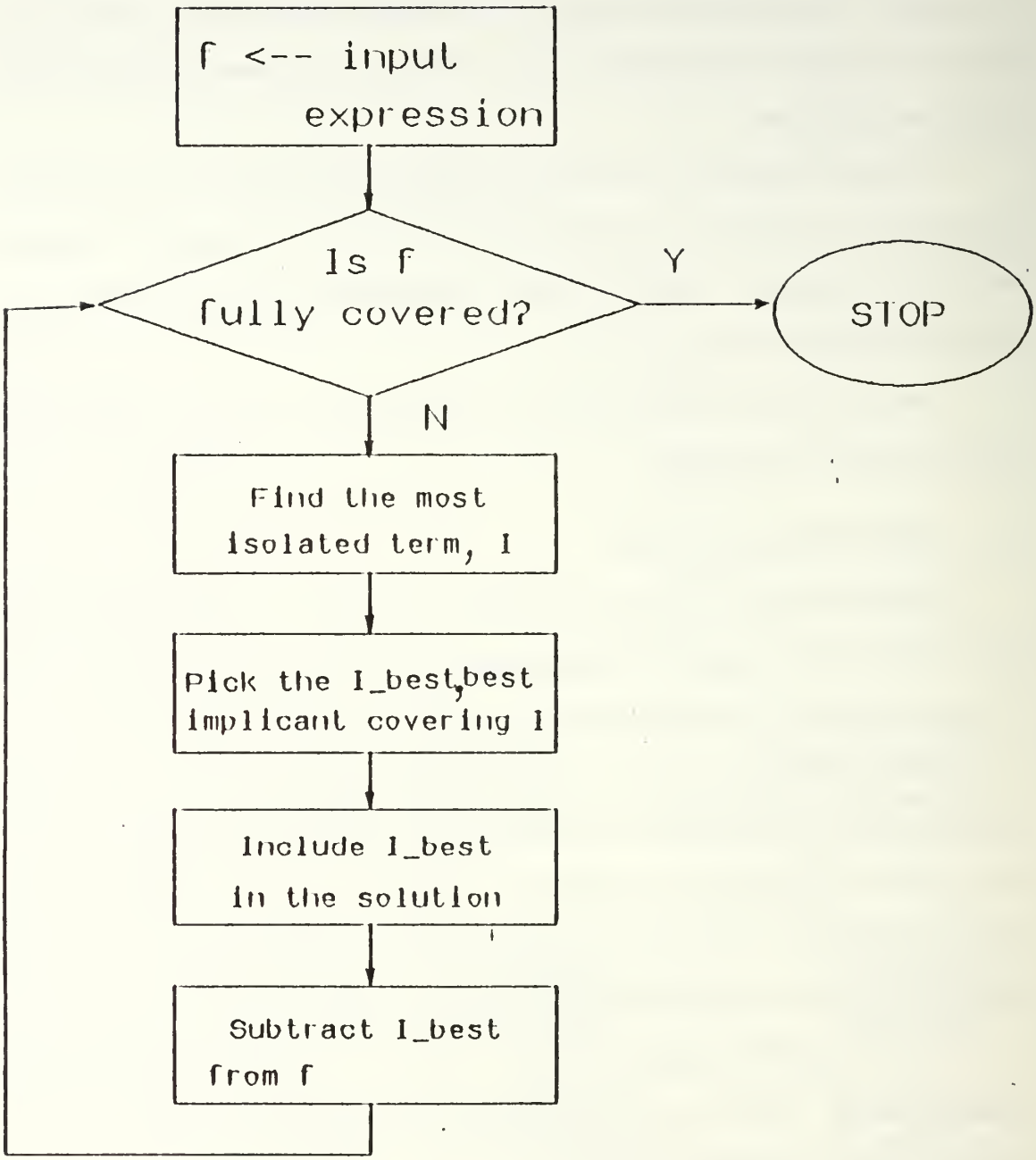

Figure 3.5: Flowchart of Logic Minimization Routine

are functions where the first does better than the second and vice versa. Thus, the technique of Gold is introduced in Reference 7. This is simply the application of both algorithms followed by a choice of the best realization. Since the CAD tool is also used to analyze minimization algorithms, implementation of all heuristics is an integral part of the program. All of them work in an identical manner except in two ways. One is the way a minterm is chosen, and the other is the way an implicant is chosen.

- Pomper and Armstrong [Ref. 5]

The minterm is chosen randomly. However, the implicant is chosen as the one which drives the most minterms to zero when subtracted. If more than one implicant is available, the largest is chosen. If there are more than one largest, the first generated is chosen.

- Dueck and Miller [Ref. 6]

In this algorithm, an isolation factor (IF) is calculated for each minterm with the smallest value beginning with all 1 minterms, 2 minterms, etc. The minterm with the largest IF is chosen and if there are more than one, the first generated is chosen. IF is calculated as the inverse of the clustering factor where the clustering factor is as follows:

$$CF_\alpha = DEA_\alpha \times (radix - 1) + EA_\alpha \tag{3.3}$$

where DEA is the number of variables (directions) in which a minterm (α) can be combined with a non-zero number of minterms (called direction of expandable adjacency). EA is the number of adjacent minterms with which a minterm (α) can be combined in an interval literal (called expandable adjacency). CF provides a

measure of the degree of which a specific minterm can combine with other minterms in the expression.

The minterm with the smallest CF is chosen as the most isolated minterm. All implicants that cover this minterm are then generated. A measure called the relative break count (RBC) is calculated for each. This provides a measure of the degree to which the expression is simplified if the implicant under consideration is chosen. The implicant with the smallest RBC is chosen as the best implicant. If there are more than one best implicant, the first generated is chosen.

- Gold [Ref. 7]

Gold is a heuristic in which the algorithms of Pomper and Armstrong and Dueck and Miller are applied, and the best realization (the one with the smallest number of product terms) is chosen. It was inspired by the observation that these algorithms display a diversity in realizations and one algorithm is not consistently better than the others over all expressions. However, certain classes of expressions do show that one algorithm does perform consistently better. With Gold, the combination of algorithms takes advantage of the best features of each.

Table 3.2 shows the comparison of three heuristic algorithms. When logic minimization is accomplished, the minimal solution of the original input expression is as shown in Figure 3.6 and Equation 3.4. Note that the number of product terms in the input expression in Equation 3.1 was 11 (eleven). Now the minimal solution (Equation 3.4) has 8 (eight) product terms in it. There is about 25% reduction in the number of product terms.

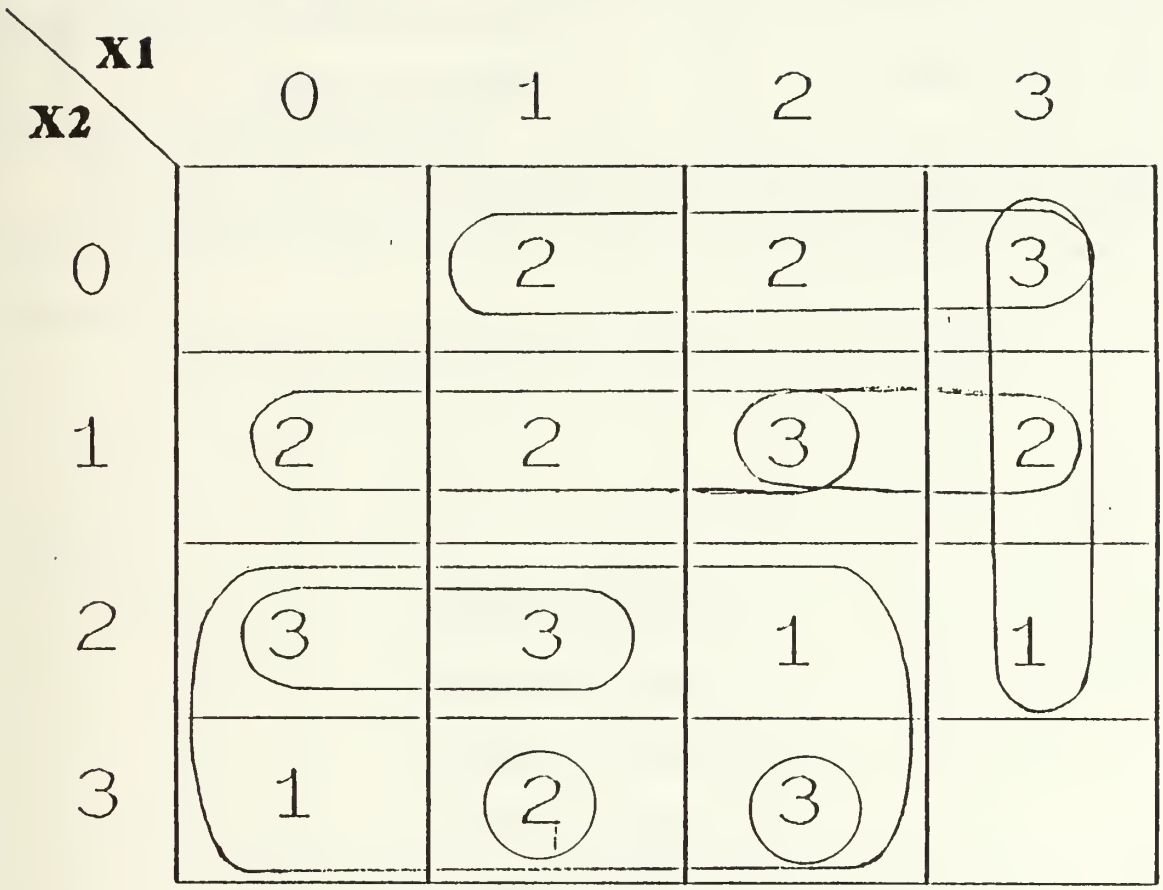

Figure 3.6: The Minimal Solution of Input Expression from Logic Minimization Routine

Algorithm	Choice of Minterm	Choice of Implicant
1. Pomper and Armstrong	Random	Drives Most Minterms to 0
2. Dueck and Miller	Largest IF (isolation factor)	Smallest RBC (relative break count)
3. Gold	Best of 1 and 2	

TABLE 3.2: Comparison of Minimization Algorithms

$$4:2: \tag{3.4}$$

$$+1 * X1(0,2) * X2(2,3)$$

$$+1 * X1(3,3) * X2(0,2)$$

$$+1 * X1(1,1) * X2(3,3)$$

$$+1 * X1(2,3) * X2(1,1)$$

$$+2 * X1(2,2) * X2(3,3)$$

$$+2 * X1(1,3) * X2(0,0)$$

$$+2 * X1(0,2) * X2(1,1)$$

$$+2 * X1(0,1) * X2(2,2)$$

C. LOGIC SIMULATION

This routine is used to verify the minimal solution from the logic minimization routine. The user can specify certain specific values or range of values for each variable in the input expression (called test vector). The logic simulation routine evaluates the minimized function value for each coordinate from the test vector. The user can do the verification test by comparing the original expression with the output of the Logic Simulation routine.

D. LAYOUT GENERATION

The output of the Logic Minimization routine is used to produce a layout file (called a magic file) for MVL-PLA chips. This routine is discussed in more detail in Reference 8.

IV. CONCLUSION

This thesis presents a CAD tool for design of multiple-valued current-mode programmable logic arrays. It was designed to be user friendly, requiring little knowledge of programming techniques. Appendix A contains a user manual written for the first time user. In the interest of clarity, details of the program have been omitted. However, for the interested user, the program has been extensively documented. This documentation and the program are contained in Appendix B. Three main conclusions have resulted from this work.

- *Conclusion 1 - Design/Analysis Tool*

The CAD tool can be used for two purposes. One is as a design tool and the other as an analysis tool. It is used for designing practical current-mode MVL CMOS circuits and also used for analyzing and comparing the different minimization algorithms. In fact, it is designed to get the statistics for each minimization program, which is very useful to obtain insight into logic minimization algorithms.

- *Conclusion 2 - Further Improvement in CAD*

Further research and improvements are needed in two specific areas. These are 1) Logic Minimization and 2) Layout Generation. Still there are cases which the heuristics give poorer realizations than supplied the user. Programs are desired which can find the minimal solution closest to the optimal solution. In MVL-PLA layout, there is some waste of the chip area. The further optimized layout configuration should be considered.

- *Conclusion 3 - Execution Speed Improvement*

The Profiler Utility shows how many times each routine is called and the percentage of time spent in executing that routine. Below is an example output from this routine. According to the profiler output shown in Table 4.1, there are several subroutines which are called most frequently throughout the program execution such as _eval, _vcopy , and _next_coord. Typical runs show these three routines as the most time consuming. If those subroutines are carefully examined and improved, the total execution time will be reduced significantly.

%time	cumsecs	#call	ms/call	name
92.0	141.83	863184	0.16	__eval
1.3	143.79	863184	0.00	_vcopy
2.1	147.08	443878	0.01	_next_coord
0.0	147.13	6133	0.00	_next_implicant
0.0	147.16	1789	0.01	_copy_implicant
0.0	147.17	391	0.03	_yylook
0.0	147.18	358	0.03	_yylex
0.0	147.18	358	0.00	_next_token
0.0	147.19	356	0.03	_strcat
0.0	147.20	244	0.04	_strcmp
0.0	147.20	244	0.00	_match
0.0	147.20	170	0.00	_strlen
0.0	147.20	164	0.00	__filbuf
0.0	147.20	162	0.00	__doscan
0.0	147.20	162	0.00	_sscanf
0.0	147.20	158	0.00	_malloc
0.0	147.20	140	0.00	_free
0.0	147.20	112	0.00	_alloc_bound
0.0	147.20	96	0.00	_limit
0.0	147.20	80	0.00	_alloc_term
0.0	147.20	80	0.00	_var_id
0.0	147.20	80	0.00	_var_list
0.0	147.20	80	0.00	_variable
0.0	147.20	77	0.00	_realloc
0.0	147.20	38	0.00	_bcopy
0.0	147.20	32	0.00	_gen_bounds
0.0	147.20	32	0.00	_init_implicant
0.0	147.20	32	0.00	_subtract_implicant
0.0	147.20	32	0.00	_term
0.0	147.20	32	0.00	_term_list

TABLE 4.1: The Output of the Profiler

APPENDIX A

Getting Started

A few examples are included in this appendix A in order to demonstrate the use of this CAD tool,the appearance of a terminal session, and the capability of the utility functions.

1. DESIGN PROCEDURES

(a) LOGGING IN

- LOG IN ON A VAX TERMINAL.

- CHANGE TO SUBDIRECTORY 'mag' IN YOUR WORKING DIRECTORY.

```
% cd mag
```

(b) CREATING THE INPUT DATA FILE

- BY USING 'vi'EDITOR, MAKE AN INPUT DATA FILE.

```
% vi tt2.mvl
```

(1) WHEN THE USER HAS A SINGLE INPUT EXPRSSION

```
4:2:
+2*X1(1,3)*X2(1,1)
+1*X1(1,3)*X2(1,3)
+2*X1(1,2)*X2(0,2)
+1*X1(1,2)*X2(3,3)
+2*X1(0,2)*X2(0,2)
+2*X1(3,3)*X2(2,3)
+3*X1(0,2)*X2(0,0)
+1*X1(1,2)*X2(0,2)
+1*X1(0,3)*X2(1,2)
+1*X1(1,2)*X2(1,2)
+1*X1(1,2)*X2(0,1);
```

(tt2.mvl is the name of input data file. In this example there is only one expression. It is a 4-valued 2-variable expression with 11 terms in it. A ;(semicolon) terminates the expression.)

(2) WHEN THE USER HAS MULTIPLE EXPRESSIONS,THE INPUT FILE MAY LOOK AS FOLLOWS. THERE ARE THREE EXPRESSIONS AND ALL OF THEM ARE 4-VALUED 2-VARIABLE LOGIC EXPRESSIONS.

```
4:2:
+2*X1(3,3)*X2(1,3)
+3*X1(1,3)*X2(1,2)
+1*X1(1,3)*X2(1,1)
+1*X1(1,3)*X2(1,3)
+2*X1(1,2)*X2(0,2);
4:2:
+1*X1(1,2)*X2(3,3)
+2*X1(0,2)*X2(0,2)
+2*X1(3,3)*X2(2,3);
4:2:
+1*X1(1,2)*X2(1,2)
```

```
+1*X1(1,2)*X2(0,1)
+2*X1(0,0)*X2(1,1)
+3*X1(0,1)*X2(1,3);
(input data file; tt3.mvl )
```

(3) WHEN THE USER APPLIES TEST VECTORS TO EACH VARIABLE FOR VERIFICATION,

THE INPUT DATA FILE LOOKS AS FOLLOWS.

```
4:2:
+2*X1(1,3)*X2(1,1)
+1*X1(1,3)*X2(1,3)
+2*X1(1,2)*X2(0,2)
+1*X1(1,2)*X2(3,3)
+2*X1(0,2)*X2(0,2)
+2*X1(3,3)*X2(2,3)
+3*X1(0,2)*X2(0,0)
+1*X1(1,2)*X2(0,2)
+1*X1(0,3)*X2(1,2)
+1*X1(1,2)*X2(1,2)
+1*X1(1,2)*X2(0,1);

X1(0,3)X2(0,3);
(test vector for the input expression above. Be sure to put ';' at the
 end. No need to put '*' between variables. In this case test vector
 has the range 0 through 3 for both variables X1 and X2. The user can
 specify any arbitrary test vector to each variable, for instance,
 X1(0,1)X2(2,2).)
 ( input data file; tt4.mvl )
```

(4) WHEN THERE ARE DON'T CARE TERMS IN THE EXPRESSION,USE '?'(QUESTION MARK)

FOR THE COEFFICIENT,AS THE FOLLOWING EXAMPLE.

```
4:2:
+2*X1(1,3)*X2(1,1)
+1*X1(1,3)*X2(1,3)
+2*X1(1,2)*X2(0,2)
+?*X1(1,2)*X2(3,3)
+2*X1(0,2)*X2(0,2)
+2*X1(3,3)*X2(2,3)
+3*X1(0,2)*X2(0,0)
+1*X1(1,2)*X2(0,2)
+1*X1(0,3)*X2(1,2)
+1*X1(1,2)*X2(1,2)
+1*X1(1,2)*X2(0,1);

X1(0,3)X2(0,3);
( input data file; tt5.mvl )
( Don't care term is 4th term from above with its coefficient '?')
```

(c) HOW TO PERFORM LOGIC MINIMIZATION AND SELECT OUTPUT OPTIONS TO GET THE

DESIRED RESULTS.

GETTING HELP:THERE IS A HELP ROUTINE FOR THE USERS WHO ARE NOT FAMILIAR

WITH ALL OPTIONS AVAILABLE IN THIS CAD.

TYPE THE FOLLOWING COMMAND FOR HELP

```
% mvl --
```

```
(The following HELP messages appears on the screen.....)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

usage: mvl [-eimrsvqDPG] [-0.0] [-oFile] [-xFile] [file ...]
    -e        - Print the original expression, as parsed
    -i        - Print each implicant found
    -m        - Print the Karnaugh map of the original and each
              successive expression
              WARNING: impractical for nvar > 3
    -v        - Verify the minimal solution
    -s        - Print statistics on selected heuristics
    -q        - Quiet, don't print final results
    -0.0      - Build source expressions for all solutions
              whose ratio to the original number of terms
              exceeds this number and output them to a file
              (default is "x.mvl")
    -D        - Execute the Dueck & Miller heuristic (default)
    -P        - Execute the Pomper & Armstrong heuristic
    -G        - Execute the Gold heuristic
    -xFile    - Output source expressions built using -0.0 to "File"
              instead of "x.mvl"
    -oFile    - Ouput formatted solutions to "File"

(Note; In the above HELP messages "print" means "output to CRT".)



(EXAMPLE 1) USING POMPER AND ARMSTRONG ALGORITHM ON tt2.mvl, OUTPUT ORIGINAL

            EXPRESSION AS PARSED AND FINAL RESULT.

(command line is as follows......)
ece:/work/mag
% mvl -Pe tt2.mvl

( the following appears on the screen.....)

MVL  V1.0  Copyright 1988  Naval Postgraduate School


Expression:

  radix:   4
  nvars:   2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3)
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
```

```
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

  P&A:     3/11   implicants - 0.27
( the last line shows that the POMPER AND ARMSTRONG heuristic produced a
  realization of 3 product terms  starting with 11 product terms,  for a
  reduction of 27%)
```

(EXAMPLE 2) USING DUECK AND MILLER ALGORITHM ON tt2.mv1, OUTPUT

EACH IMPLICANT FOUND AND FINAL RESULT.

```
(command line is as follows ... )
ece:/work/mag
% mvl -Di tt2.mvl

( the following appears on the screen .....)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

  D&M MIM:  2*X1( 1, 1)*X2( 3, 3)
      Imp:  2*X1( 1, 2)*X2( 3, 3)

  D&M MIM:  3*X1( 3, 3)*X2( 3, 3)
      Imp:  3*X1( 3, 3)*X2( 1, 3)

  D&M MIM:  3*X1( 2, 2)*X2( 0, 0)
      Imp:  3*X1( 0, 2)*X2( 0, 2)

  D&M:     3/11   implicants - 0.27
```

(EXAMPLE 3) USING GOLD ALGORITHM, OUTPUT THE KARNAUGH MAP OF THE ORIGINAL AND

EACH SUCCESSIVE EXPRESSION AND FINAL RESULT.

```
(command line is as follows ...)
ece:/work/mag
% mvl -Gm tt2.mvl

( the following appears on the screen......)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

 Orig map (D&M):
  3.  3.  3.  0
```

```
  3.   3.   3.   3.
  3.   3.   3.   3.
  0    2    2    3.

  3.   3.   3.   0
  3.   3.   3.   3.
  3.   3.   3.   3.
  0    0    0    3.

  3.   3.   3.   0
  3.   3.   3.   4.
  3.   3.   3.   4.
  0    0    0    4.

  4.   4.   4.   0
  4.   4.   4.   4.
  4.   4.   4.   4.
  0    0    0    4.

 Orig map (P&A):
  3.   3.   3.   0
  3.   3.   3.   3.
  3.   3.   3.   3.
  0    2    2    3.

  4.   4.   4.   0
  4.   4.   4.   3.
  4.   4.   4.   3.
  0    2    2    3.

  4.   4.   4.   0
  4.   4.   4.   4.
  4.   4.   4.   4.
  0    2    2    4.

  4.   4.   4.   0
  4.   4.   4.   4.
  4.   4.   4.   4.
  0    0    0    4.

  Gold(===):    3/11   implicants - 0.27
```

(Note; .(dot) after a number denotes a highest logic value or don't care generated in the course of a minimization. Don't cares in the original function have no dot.)

(EXAMPLE 4) USING POMPER AND ARMSTRONG ON tt2.mvl, OUTPUT THE ORIGINAL EXPRESSION AND K-MAP OF THE ORIGINAL EXPERSSION AND EACH SUCCESSIVE EXPRESSION WITH FINAL RESULT.

(commnad line is as follows .....)

```
ece:/work/mag
% mvl -Pei tt2.mvl
```

(the following appears on the screen.....)

```
MVL  V1.0  Copyright 1988  Naval Postgraduate School
```

```
Expression:

  radix:   4
  nvars:   2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3)
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

  P&A MIM:  3*X1( 1, 1)*X2( 0, 0)
      Imp:  3*X1( 0, 2)*X2( 0, 2)

  P&A MIM:  3*X1( 3, 3)*X2( 3, 3)
      Imp:  3*X1( 3, 3)*X2( 1, 3)

  P&A MIM:  2*X1( 1, 1)*X2( 3, 3)
      Imp:  2*X1( 1, 2)*X2( 0, 3)

  P&A:    3/11   implicants - 0.27
```

(EXAMPLE 5) USING DUECK AND MILLER ON tt2.mvl, OUTPUT THE ORIGINAL EXPRESSION AND K-MAP WITH FINAL RESULT.

```
(command line is as follows......)
ece:/work/mag
% mvl -Dem tt2.mvl

(the following appears on the screen.......)

MVL  V1.0  Copyright 1988  Naval Postgraduate School
```

```
Expression:

  radix:   4
  nvars:   2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3)
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

 Orig map (D&M):
  3.  3.  3.  0
  3.  3.  3.  3.
  3.  3.  3.  3.
  0   2   2   3.

  3.  3.  3.  0
  3.  3.  3.  3.
  3.  3.  3.  3.
  0   0   0   3.

  3.  3.  3.  0
  3.  3.  3.  4.
  3.  3.  3.  4.
  0   0   0   4.

  4.  4.  4.  0
  4.  4.  4.  4.
  4.  4.  4.  4.
  0   0   0   4.

  D&M:    3/11   implicants - 0.27
```

(EXAMPLE 6) USING GOLD ALGORITHM ON tt2.mvl, OUTPUT THE ORIGINAL EXPRESSION, K-MAP AND EACH IMPLICANT FOUND WITH FINAL RESULT.

```
( command line is as follows......)
ece:/work/mag
% mvl -Geim tt2.mvl

( The following appears on the screen.......)

MVL  V1.0  Copyright 1988  Naval Postgraduate School


Expression:

  radix:   4
  nvars:   2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3).
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

 Orig map (D&M):
  3.  3.  3.  0
  3.  3.  3.  3.
  3.  3.  3.  3.
  0   2   2   3.

  D&M MIM:  2*X1( 1, 1)*X2( 3, 3)
      Imp:  2*X1( 1, 2)*X2( 3, 3)

  3.  3.  3.  0
  3.  3.  3.  3.
  3.  3.  3.  3.
  0   0   0   3.
```

```
   D&M MIM:   3*X1( 3, 3)*X2( 3, 3)
       Imp:   3*X1( 3, 3)*X2( 1, 3)

   3.   3.   3.   0
   3.   3.   3.   4.
   3.   3.   3.   4.
   0    0    0    4.

   D&M MIM:   3*X1( 2, 2)*X2( 0, 0)
       Imp:   3*X1( 0, 2)*X2( 0, 2)

   4.   4.   4.   0
   4.   4.   4.   4.
   4.   4.   4.   4.
   0    0    0    4.


Expression:

  radix:    4
  nvars:    2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3)
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

Orig map (P&A):
 3.   3.   3.   0
 3.   3.   3.   3.
 3.   3.   3.   3.
 0    2    2    3.

 P&A MIM:   3*X1( 1, 1)*X2( 0, 0)
     Imp:   3*X1( 0, 2)*X2( 0, 2)
```

```
  4.  4.  4.  0
  4.  4.  4.  3.
  4.  4.  4.  3.
  0   2   2   3.

  P&A MIM:  3*X1( 3, 3)*X2( 3, 3)
      Imp:  3*X1( 3, 3)*X2( 1, 3)

  4.  4.  4.  0
  4.  4.  4.  4.
  4.  4.  4.  4.
  0   2   2   4.

  P&A MIM:  2*X1( 1, 1)*X2( 3, 3)
      Imp:  2*X1( 1, 2)*X2( 0, 3)

  4.  4.  4.  0
  4.  4.  4.  4.
  4.  4.  4.  4.
  0   0   0   4.

  Gold(===):    3/11   implicants - 0.27
(Note; Gold(===) means that both P&A and D&M have the same result and
 the resulting expression of D&M is chosen on ties.)
```

(EXAMPLE 7) USING GOLD ON tt2.mvl, OUTPUT STATISTICS ON SELECTED ALGORITHM WITH FINAL RESULT.

```
(command line is as follows.......)
ece:/work/mag
% mvl -Gs tt2.mvl

(The following appears on the screen......)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(===):    3/11   implicants - 0.27

Statistics for D&M:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
   4     2        1       11      11.00

                  Eval      Comp      Pick      Comp      Gen       Next      Valid
                  Expr      CF        MIM       RBC       Bounds    Impl      Impl
Tot:              1254      173         4       23          3       27        24
Avg/Expr:      1254.00   173.00      4.00    23.00       3.00    27.00     24.00
Avg/Term:       114.00    15.73      0.36     2.09       0.27     2.45      2.18

Statistics for P&A:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
   4     2        1       11      11.00

                  Eval      Pick      Gen       Next      Valid
                  Expr      MIM       Bounds    Impl      Impl
Tot:              534         4         3       30        27
Avg/Expr:      534.00      4.00      3.00    30.00     27.00
Avg/Term:       48.55      0.36      0.27     2.73      2.45
```

(EXAMPLE 8) USING GOLD ON tt4.mvl, VERIFY THE MINIMAL SOLUTION BY TEST VECTORS SPECIFIED IN THE INPUT FILE.

```
(command line is as follows......)
ece:/work/mag
% mvl -Gv tt4.mvl

( the following appears on the screen.....)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(===):    3/11   implicants - 0.27

Verification:

  X1(0,3)
  X2(0,3)

  (0,0) =   3
  (1,0) =   3
  (2,0) =   3
  (3,0) =   0
  (0,1) =   3
  (1,1) =   3
  (2,1) =   3
  (3,1) =   3
  (0,2) =   3
  (1,2) =   3
  (2,2) =   3
  (3,2) =   3
  (0,3) =   0
  (1,3) =   2
  (2,3) =   2
  (3,3) =   3
```

(EXAMPLE 9) USING GOLD ON tt5.mvl,OUTPUT THE ORIGINAL EXPRESSION,IMPLICANT FOUND, K-MAP, STATISTICS ON THE SELECTED ALGORITHM,AND VERIFICATION OF FINAL RESULT.

INPUT EXPERSSION HAS A DON'T CARE TERM IN IT.

```
(command line is as follows......)
ece:/work/mag
% mvl -Geimsv tt5.mvl

(The following appears on the screen.....)

MVL  V1.0  Copyright 1988  Naval Postgraduate School


Expression:

  radix:   4
  nvars:   2
```

```
nterms: 11
coeff: 2
  X1(1,3)
  X2(1,1)
coeff: 1
  X1(1,3)
  X2(1,3)
coeff: 2
  X1(1,2)
  X2(0,2)
coeff: 4
  X1(1,2)
  X2(3,3)
coeff: 2
  X1(0,2)
  X2(0,2)
coeff: 2
  X1(3,3)
  X2(2,3)
coeff: 3
  X1(0,2)
  X2(0,0)
coeff: 1
  X1(1,2)
  X2(0,2)
coeff: 1
  X1(0,3)
  X2(1,2)
coeff: 1
  X1(1,2)
  X2(1,2)
coeff: 1
  X1(1,2)
  X2(0,1)

Orig map (D&M):
 3.  3.  3.  0
 3.  3.  3.  3.
 3.  3.  3.  3.
 0   4   4   3.

 D&M MIM:  3*X1( 3, 3)*X2( 1, 1)
     Imp:  3*X1( 0, 3)*X2( 1, 2)

 3.  3.  3.  0
 4.  4.  4.  4.
 4.  4.  4.  4.
 0   4   4   3.

 D&M MIM:  3*X1( 3, 3)*X2( 3, 3)
     Imp:  3*X1( 3, 3)*X2( 3, 3)

 3.  3.  3.  0
 4.  4.  4.  4.
 4.  4.  4.  4.
 0   4   4   4.

 D&M MIM:  3*X1( 2, 2)*X2( 0, 0)
     Imp:  3*X1( 0, 2)*X2( 0, 0)

 4.  4.  4.  0
 4.  4.  4.  4.
 4.  4.  4.  4.
 0   4   4   4.
```

```
Expression:

  radix:   4
  nvars:   2
  nterms: 11
  coeff: 2
    X1(1,3)
    X2(1,1)
  coeff: 1
    X1(1,3)
    X2(1,3)
  coeff: 2
    X1(1,2)
    X2(0,2)
  coeff: 4
    X1(1,2)
    X2(3,3)
  coeff: 2
    X1(0,2)
    X2(0,2)
  coeff: 2
    X1(3,3)
    X2(2,3)
  coeff: 3
    X1(0,2)
    X2(0,0)
  coeff: 1
    X1(1,2)
    X2(0,2)
  coeff: 1
    X1(0,3)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(1,2)
  coeff: 1
    X1(1,2)
    X2(0,1)

 Orig map (P&A):
  3.  3.  3.  0
  3.  3.  3.  3.
  3.  3.  3.  3.
  0   4   4   3.

  P&A MIM:  3*X1( 2, 2)*X2( 1, 1)
      Imp:  3*X1( 0, 2)*X2( 0, 2)

  4.  4.  4.  0
  4.  4.  4.  3.
  4.  4.  4.  3.
  0   4   4   3.

  P&A MIM:  3*X1( 3, 3)*X2( 2, 2)
      Imp:  3*X1( 1, 3)*X2( 1, 3)

  4.  4.  4.  0
  4.  4.  4.  4.
  4.  4.  4.  4.
  0   4   4   4.

  Gold(P&A):    3/11   implicants - 0.27

Verification:

  X1(0,3)
```

```
X2(0,3)

(0,0) =   3
(1,0) =   3
(2,0) =   3
(3,0) =   0
(0,1) =   3
(1,1) =   3
(2,1) =   3
(3,1) =   3
(0,2) =   3
(1,2) =   3
(2,2) =   3
(3,2) =   3
(0,3) =   0
(1,3) =   3
(2,3) =   3
(3,3) =   3

Statistics for D&M:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
   4     2        1        11      11.00

                  Eval     Comp      Pick      Comp      Gen       Next      Valid
                  Expr     CF        MIM       RBC       Bounds    Impl      Impl
Tot:              1334     169       4         31        3         36        33
Avg/Expr:       1334.00   169.00     4.00      31.00     3.00      36.00     33.00
Avg/Term:        121.27    15.36     0.36       2.82     0.27       3.27      3.00

Statistics for P&A:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
   4     2        1        11      11.00

                  Eval      Pick      Gen       Next      Valid
                  Expr      MIM       Bounds    Impl      Impl
Tot:              811       3         2         54        52
Avg/Expr:       811.00      3.00      2.00      54.00     52.00
Avg/Term:        73.73      0.27      0.18       4.91      4.73
```

(EXAMPLE 10) USING GOLD ON tt2.mvl, IF USER DOESN'T NEED FINAL RESULT

SPECIFY -q FLAG.

(command line is as follows.....)

```
ece:/work/mag
% mvl -Gq tt2.mvl
```

(the following appears on the screen.....)

```
MVL  V1.0  Copyright 1988  Naval Postgraduate School
```

(d) HOW TO GENERATE LAYOUT FILE IN MAGIC FORMAT.

```
    SPECIFY INPUT , OUTPUT FILE AND THE SELECTED MINIMIZATION ALGORITHM.

    THE FINAL RESULT  WILL  BE  STORED IN THE OUTPUT FILE WITH FORMATTED

    SOLUTION.

(command line is as follows. input file: tt2.mvl, outfile: tt2.out
 and output file should have -o flag.)
ece:/work/mag
% mvl -G tt2.mvl -ott2.out

( The formatted final solution will be saved in an output file which
  will be used for layout generation. If the minimization result is
  worse than the original expression, the original expression will be
  output for producing the layout.)

( the following appears on the screen.....)
MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(===):    3/11   implicants - 0.27


(The formatted output is as follows.)

# of input   # of output
coeff
lower upper
lower upper
...
99 (sentinel)

  /* # of input = # of variables
     # of output = # of expressions */


(an example of the formatted output)

2 1
2
1 2
3 3
3
3 3
1 3
3
0 2
0 2
99

( The following command initiates the layout generation
  by using output file ,called tt2.out )
ece:/work/mag
% mvpla tt2.out


(The following appears on the screen .....)


**********************************************

         Input       : tt2.out
         Output file : tt2.out.mag

**********************************************
```

```
Check:
 2 inputs 1 outputs function


f0 = 2x0(1,2)x1(3,3)
   + 3x0(3,3)x1(1,3)
   + 3x0(0,2)x1(0,2)


Now, generating CMOS-MV-PLA layout in magic format......

f0 = 2x0(1,2)x1(3,3)
   + 3x0(3,3)x1(1,3)
   + 3x0(0,2)x1(0,2)
Output to tt2.out.mag...


Done !

************* EXIT *************


( Once layout file with the Magic format is created,
  the user needs to invoke the Magic graphic package.)
( command line is as follows.....)
ece:/work/mag
% Magic tt2.out


(The following appears on the screen .....)

Magic - Version 4.10 - Last updated 12/6/85 at 18:33:06

Using technology "scmos".

tt2.out: 500 rects



(After the whole layout is displayed on the CRT, the user can create a file
 for use  in  generating a hard  copy  of the layout and exit by typing the
 the following command to create the .cif file for plotting.)

:cif write tt2.out
>:quit


(The following command line allows the user to have a hard copy of layout.)
ece:/work/mag
% Cifplot -r tt2.out.cif
[1] 20267

ece:/work/mag
% Window: -10950 12600 -2100 51000
Scale: 1 micron is 0.0198588 inches (504x)
The plot will be 0.39 feet

[1]    Done                    cifplot -I -P /tools/berk86/lib/patterns -r tt2.out.


(the following is an example of a hard copy of MVL-PLA layout.)
```

cifplot: Window: -18950 12600 -2100 15000 --- Scale: 1 micron is 0.0196588 inches (504x)

****
****
****

THIS COMPLETES THE DESIGN PROCEDURE FOR THIS CAD TOOL.

****
****
****

2.ANALYSIS PROCEDURES

THERE IS A RANDOM EXPRESSION GENERATOR IN THIS CAD. THE USER CAN GENERATE A SET OF RANDOM EXPRESSIONS OF A SPECIFIC CLASS AND ANALYZE THE DIFFERENT MINIMIZATION PROGRAMS.

(A) HOW TO GENERATE RANDOM EXPRESSIONS

(EXAMPLE 1) THERE IS HELP MANUAL FOR USERS WHICH CAN BE OBTAINED AS FOLLOWS.

```
ece:/work/mag
% mvltest --
```

(The following HELP messages appears on the screen.....)

```
MVLTEST  V1.0  Copyright 1988 Naval Postgraduate School

usage: mvltest [-sN -rN -iN -tN -oN]
    -sN   - Cycle the random seed N times (default = 0)
    -rN   - Set radix to N (default = 4)
    -iN   - Set number of inputs to N (default = 2)
    -tN   - Set number of terms to N (default = 1)
    -oN   - Set number of output to N (default = 1)
```

(EXAMPLE 2) THE FOLLOWING COMMAND INITIATES A RANDOM FUNCTION GENERATOR TO CREATE A SET OF RANDOM EXPRESSIONS. IT CYCLES THE RANDOM SEED 50 TIMES AND GENERATE 20 EXPRESSIONS OF 4-VALUED 2-VARIABLE LOGIC EXPRESSION. ALL OF THEM ARE STORED IN tt6.mvl AND HAVE 10 PRODUCT TERMS. THE NUMBER OF OUTPUTS IS THE NUMBER OF FUNCTIONS IN THE SAMPLE SET.

(command line is as follows.!...)

```
ece:/work/mag
% mvltest -s50 -r4 -i2 -t10 -o20 > tt6.mvl
```

(The following is a set of random expressions generated by the previous command.)

```
4:2:
   +2*X1(1,3)*X2(3,3)
   +1*X1(0,0)*X2(1,2)
   +1*X1(0,3)*X2(1,2)
   +1*X1(1,2)*X2(1,2)
   +1*X1(1,2)*X2(0,1)
   +2*X1(0,0)*X2(1,1)
   +3*X1(0,1)*X2(1,3)
   +2*X1(3,3)*X2(0,1)
   +3*X1(0,3)*X2(1,1)
   +2*X1(0,3)*X2(0,0);
4:2:
   +1*X1(0,3)*X2(0,3)
   +2*X1(1,1)*X2(0,3)
   +1*X1(1,2)*X2(0,1)
   +1*X1(1,1)*X2(0,2)
   +3*X1(0,0)*X2(2,2)
   +1*X1(3,3)*X2(0,3)
   +2*X1(1,2)*X2(1,1)
   +2*X1(1,1)*X2(2,2)
   +3*X1(3,3)*X2(0,3)
   +1*X1(1,1)*X2(1,1);
4:2:
   +2*X1(1,2)*X2(0,1)
   +3*X1(1,2)*X2(0,3)
   +3*X1(2,3)*X2(0,1)
```

```
    +2*X1(0,1)*X2(2,3)
    +2*X1(1,3)*X2(2,2)
    +1*X1(0,2)*X2(3,3)
    +3*X1(1,1)*X2(0,1)
    +1*X1(1,2)*X2(0,0)
    +2*X1(0,0)*X2(0,3)
    +3*X1(3,3)*X2(0,2);
4:2:
    +2*X1(1,2)*X2(3,3)
    +2*X1(1,2)*X2(2,3)
    +1*X1(0,1)*X2(1,2)
    +1*X1(1,3)*X2(3,3)
    +3*X1(0,3)*X2(1,1)
    +1*X1(3,3)*X2(2,3)
    +3*X1(3,3)*X2(1,2)
    +1*X1(1,2)*X2(3,3)
    +1*X1(0,1)*X2(3,3)
    +3*X1(0,1)*X2(1,3);
4:2:
    +2*X1(3,3)*X2(2,2)
    +3*X1(2,2)*X2(3,3)
    +1*X1(0,1)*X2(1,3)
    +3*X1(1,3)*X2(2,3)
    +2*X1(1,1)*X2(0,2)
    +3*X1(0,2)*X2(2,3)
    +1*X1(2,2)*X2(1,1)
    +1*X1(2,2)*X2(1,2)
    +1*X1(2,2)*X2(2,2)
    +3*X1(0,1)*X2(1,2);
4:2:
    +3*X1(1,2)*X2(1,2)
    +3*X1(2,3)*X2(1,1)
    +1*X1(2,2)*X2(0,0)
    +3*X1(1,2)*X2(1,1)
    +1*X1(3,3)*X2(1,2)
    +3*X1(1,3)*X2(0,0)
    +2*X1(0,2)*X2(1,2)
    +1*X1(0,1)*X2(1,3)
    +1*X1(0,1)*X2(0,0)
    +3*X1(3,3)*X2(0,0);
4:2:
    +3*X1(0,3)*X2(1,3)
    +3*X1(1,1)*X2(1,3)
    +2*X1(0,3)*X2(0,2)
    +3*X1(2,3)*X2(0,1)
    +2*X1(0,1)*X2(3,3)
    +3*X1(0,2)*X2(3,3)
    +3*X1(1,1)*X2(0,3)
    +3*X1(1,1)*X2(1,1)
    +2*X1(0,1)*X2(1,1)
    +3*X1(2,2)*X2(0,3);
4:2:
    +3*X1(1,2)*X2(1,2)
    +1*X1(0,3)*X2(2,3)
    +2*X1(3,3)*X2(2,3)
    +3*X1(0,3)*X2(1,1)
    +2*X1(2,3)*X2(2,3)
    +2*X1(0,0)*X2(0,0)
    +3*X1(0,0)*X2(1,1)
    +3*X1(0,1)*X2(1,2)
    +3*X1(2,3)*X2(0,0)
    +3*X1(2,3)*X2(0,1);
4:2:
    +3*X1(2,3)*X2(0,3)
    +3*X1(0,2)*X2(0,2)
    +3*X1(1,2)*X2(2,2)
```

```
    +1*X1(0,0)*X2(2,3)
    +1*X1(0,3)*X2(2,2)
    +2*X1(1,3)*X2(0,0)
    +2*X1(1,1)*X2(1,3)
    +1*X1(0,2)*X2(0,1)
    +3*X1(0,1)*X2(3,3)
    +2*X1(2,2)*X2(1,1);
4:2:
    +3*X1(2,2)*X2(0,1)
    +1*X1(0,2)*X2(2,2)
    +3*X1(0,3)*X2(2,3)
    +2*X1(0,1)*X2(1,3)
    +2*X1(0,0)*X2(1,2)
    +1*X1(3,3)*X2(0,3)
    +2*X1(1,3)*X2(2,3)
    +1*X1(0,3)*X2(0,0)
    +2*X1(3,3)*X2(1,3)
    +3*X1(0,0)*X2(1,1);
4:2:
    +1*X1(0,3)*X2(2,2)
    +3*X1(1,3)*X2(0,3)
    +1*X1(2,2)*X2(0,2)
    +1*X1(1,1)*X2(0,1)
    +1*X1(1,3)*X2(1,1)
    +1*X1(3,3)*X2(2,3)
    +3*X1(2,3)*X2(2,3)
    +2*X1(0,1)*X2(1,1)
    +1*X1(1,3)*X2(0,2)
    +1*X1(0,0)*X2(3,3);
4:2:
    +1*X1(0,1)*X2(2,3)
    +2*X1(1,3)*X2(0,2)
    +3*X1(1,3)*X2(2,3)
    +1*X1(1,3)*X2(2,3)
    +1*X1(0,2)*X2(0,1)
    +3*X1(0,1)*X2(0,0)
    +2*X1(2,2)*X2(1,3)
    +3*X1(2,2)*X2(0,3)
    +1*X1(1,3)*X2(0,0)
    +1*X1(0,1)*X2(0,2);
4:2:
    +1*X1(0,3)*X2(3,3)
    +2*X1(0,3)*X2(2,2)
    +1*X1(0,3)*X2(0,3)
    +1*X1(0,1)*X2(0,2)
    +1*X1(2,3)*X2(0,3)
    +3*X1(1,3)*X2(0,2)
    +2*X1(0,1)*X2(0,2)
    +1*X1(2,2)*X2(3,3)
    +1*X1(1,1)*X2(0,3)
    +3*X1(1,2)*X2(2,2);
4:2:
    +2*X1(0,0)*X2(0,0)
    +1*X1(2,3)*X2(1,3)
    +3*X1(2,3)*X2(0,3)
    +1*X1(3,3)*X2(0,3)
    +2*X1(3,3)*X2(1,2)
    +3*X1(0,0)*X2(2,2)
    +2*X1(0,1)*X2(0,2)
    +2*X1(2,3)*X2(1,3)
    +2*X1(1,1)*X2(1,2)
    +2*X1(0,1)*X2(3,3);
4:2:
    +2*X1(1,3)*X2(1,1)
    +2*X1(0,2)*X2(0,3)
    +1*X1(0,3)*X2(3,3)
```

```
   +2*X1(0,3)*X2(0,0)
   +3*X1(0,2)*X2(0,3)
   +1*X1(1,3)*X2(1,1)
   +1*X1(0,0)*X2(1,3)
   +3*X1(2,2)*X2(0,0)
   +3*X1(2,3)*X2(1,2)
   +2*X1(0,3)*X2(0,2);
4:2:
   +2*X1(2,3)*X2(2,2)
   +2*X1(1,2)*X2(1,1)
   +3*X1(1,3)*X2(1,2)
   +2*X1(3,3)*X2(1,1)
   +1*X1(1,1)*X2(3,3)
   +3*X1(0,3)*X2(0,1)
   +3*X1(2,3)*X2(1,3)
   +1*X1(3,3)*X2(2,2)
   +1*X1(1,2)*X2(1,3)
   +2*X1(2,2)*X2(2,3);
4:2:
   +3*X1(1,2)*X2(0,0)
   +3*X1(0,1)*X2(3,3)
   +3*X1(2,3)*X2(2,2)
   +2*X1(3,3)*X2(3,3)
   +1*X1(0,0)*X2(1,3)
   +1*X1(1,1)*X2(2,2)
   +1*X1(0,3)*X2(1,3)
   +3*X1(0,1)*X2(1,1)
   +3*X1(0,3)*X2(1,2)
   +2*X1(3,3)*X2(2,2);
4:2:
   +3*X1(0,3)*X2(0,0)
   +1*X1(0,2)*X2(1,1)
   +3*X1(0,1)*X2(2,2)
   +1*X1(2,2)*X2(1,1)
   +2*X1(3,3)*X2(1,1)
   +3*X1(0,3)*X2(1,2)
   +1*X1(1,1)*X2(2,3)
   +3*X1(1,1)*X2(0,2)
   +3*X1(1,3)*X2(0,3)
   +1*X1(3,3)*X2(2,3);
4:2:
   +3*X1(0,3)*X2(0,0)
   +1*X1(1,3)*X2(0,2)
   +3*X1(2,3)*X2(0,1)
   +2*X1(0,1)*X2(1,1)
   +1*X1(0,2)*X2(0,0)
   +1*X1(1,1)*X2(0,1)
   +2*X1(3,3)*X2(3,3)
   +2*X1(0,3)*X2(3,3)
   +3*X1(1,3)*X2(1,1)
   +2*X1(0,2)*X2(0,3);
4:2:
   +3*X1(2,2)*X2(1,3)
   +1*X1(2,3)*X2(2,2)
   +1*X1(0,0)*X2(1,2)
   +2*X1(0,1)*X2(0,3)
   +3*X1(0,3)*X2(0,0)
   +2*X1(1,1)*X2(1,2)
   +3*X1(0,3)*X2(0,2)
   +3*X1(2,3)*X2(0,0)
   +1*X1(0,2)*X2(0,2)
   +2*X1(1,3)*X2(0,3);
(data file; tt6.mvl)
```

(EXAMPLE 3) THE FOLLOWING COMMAND LINE APPLIES Gold ON tt6.mvl. THE STATISTICS AND THE FINAL RESULTS ARE FOLLOWED.

```
ece:/work/mag
% mvl -Gs tt6.mvl

MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(P&A):    6/10   implicants - 0.60

  Gold(===):    6/10   implicants - 0.60

  Gold(===):    3/10   implicants - 0.30

  Gold(===):    5/10   implicants - 0.50

  Gold(===):    4/10   implicants - 0.40

  Gold(D&M):    4/10   implicants - 0.40

  Gold(===):    3/10   implicants - 0.30

  Gold(===):    4/10   implicants - 0.40

  Gold(===):    1/10   implicants - 0.10

  Gold(===):    7/10   implicants - 0.70

  Gold(===):    3/10   implicants - 0.30

  Gold(P&A):    6/10   implicants - 0.60

  Gold(===):    3/10   implicants - 0.30

  Gold(===):    5/10   implicants - 0.50

  Gold(===):    3/10   implicants - 0.30

  Gold(===):    4/10   implicants - 0.40

  Gold(D&M):    4/10   implicants - 0.40

  Gold(===):    2/10   implicants - 0.20

  Gold(===):    5/10   implicants - 0.50

  Gold(===):    3/10   implicants - 0.30

Statistics for D&M:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
   4     2       20       200       10.00

                  Eval      Comp      Pick      Comp      Gen       Next      Valid
                  Expr      CF        MIM       RBC       Bounds    Impl      Impl
Tot:             31979     3012       101       943        81       904       970
Avg/Expr:       1598.95   150.60      5.05     47.15      4.05     45.20     48.50
Avg/Term:        159.89    15.06      0.51      4.71      0.40      4.52      4.85

Statistics for P&A:

       Num     Num      Num       Avg Num
Radix  Var     Expr     Terms     Term/Expr
```

```
  4      2       20       200      10.00

                 Eval      Pick      Gen       Next      Valid
                 Expr      MIM       Bounds    Impl      Impl
Tot:            16638      101         81      1032      1033
Avg/Expr:       831.90      5.05       4.05     51.60     51.65
Avg/Term:        83.19      0.51       0.40      5.16      5.17
```

(EXAMPLE 4) HERE GENERATE ANOTHER SET OF RANDOM EXPRESSIONS,10 EXPRESSIONS OF 4-VALUED 3-VARIABLE LOGIC EXPRESSION AND SAVE THEM IN THE DATA FILE tt7.mvl.

```
(command line is as follows......)
ece:/work/mag
% mvltest -r4 -i3 -t5 -o10 > tt7.mvl
```

(EXAMPLE 5) APPLY Gold ON tt7.mvl, GET THE FINAL RESULTS AND STATISTICS.

```
(commmand line is as follows.....)
ece:/work/mag
% mvl -sG tt7.mvl

(the following appears on the screen.......)

MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(===):    4/7    implicants - 0.57

  Gold(D&M):    6/7    implicants - 0.86

  Gold(===):    7/7    implicants - 1.00

  Gold(P&A):    8/7    implicants - 1.14

  Gold(D&M):    7/7    implicants - 1.00

  Gold(D&M):    5/7    implicants - 0.71

  Gold(D&M):    6/7    implicants - 0.86

  Gold(D&M):    7/7    implicants - 1.00

  Gold(===):    7/7    implicants - 1.00

  Gold(===):    7/7    implicants - 1.00

 Statistics for D&M:

        Num       Num       Num       Avg Num
Radix   Var       Expr      Terms     Term/Expr
  4      3         10        70         7.00

                 Eval      Comp      Pick      Comp      Gen       Next      Valid
                 Expr      CF        MIM       RBC       Bounds    Impl      Impl
Tot:            54192     2525        74      1126        64       922      1257
Avg/Expr:       5419.20    252.50      7.40    112.60      6.40     92.20    125.70
Avg/Term:        774.17     36.07      1.06     16.09      0.91     13.17     17.96
```

```
Statistics for P&A:

       Num      Num       Num       Avg Num
Radix  Var      Expr      Terms     Term/Expr
   4     3        10        70         7.00

                  Eval      Pick      Gen       Next      Valid
                  Expr      MIM       Bounds    Impl      Impl
Tot:             25434       79        69      1269      1333
Avg/Expr:       2543.40     7.90      6.90    126.90    133.30
Avg/Term:        363.34     1.13      0.99     18.13     19.04
```

(EXAMPLE 6) OUTPUT SOURCE EXPRESSIONS FOR ALL SOLUTIONS WHOSE RATIO TO THE ORIGINAL NUMBER OF TERMS EXCEEDS 1.00. ALL WILL BE STORED IN tt7.out WHICH HAS -x FLAG ON IT.

(command line is as follows.....)

```
ece:/work/mag
% mvl -1.0 -xtt7.out -Gs tt7.mvl
```

(the following appears on the screen.....)

```
MVL  V1.0  Copyright 1988  Naval Postgraduate School

  Gold(===):     4/7    implicants - 0.57

  Gold(D&M):     6/7    implicants - 0.86

  Gold(===):     7/7    implicants - 1.00

  Gold(P&A):     8/7    implicants - 1.14

  Gold(D&M):     7/7    implicants - 1.00

  Gold(D&M):     5/7    implicants - 0.71

  Gold(D&M):     6/7    implicants - 0.86

  Gold(D&M):     7/7    implicants - 1.00

  Gold(===):     7/7    implicants - 1.00

  Gold(===):     7/7    implicants - 1.00

Statistics for D&M:

       Num      Num       Num       Avg Num
Radix  Var      Expr      Terms     Term/Expr
   4     3        10        70         7.00

                  Eval      Comp      Pick      Comp      Gen       Next      Valid
                  Expr      CF        MIM       RBC       Bounds    Impl      Impl
Tot:             54192     2525        74      1126        64       922      1257
Avg/Expr:       5419.20   252.50      7.40    112.60      6.40     92.20    125.70
Avg/Term:        774.17    36.07      1.06     16.09      0.91     13.17     17.96

Statistics for P&A:

       Num      Num       Num       Avg Num
```

```
Radix  Var     Expr     Terms    Term/Expr
   4     3       10        70         7.00

                  Eval      Pick      Gen       Next      Valid
                  Expr      MIM       Bounds    Impl      Impl
Tot:             25434       79        69      1269      1333
Avg/Expr:       2543.40     7.90      6.90    126.90    133.30
Avg/Term:        363.34     1.13      0.99     18.13     19.04
```

(EXAMPLE 7) THE FOLLOWING SHOWS ALL THE EXPRESSIONS WHICH EXCEEDS THE SPECIFIED RATIO IN THE PREVIOUS COMMAND LINE.

```
# P&A:8/7
4:3:
+2*x1(1,2)*x2(1,2)*x3(0,1)
    +2*x1(0,1)*x2(1,3)*x3(1,3)
    +1*x1(0,0)*x2(1,1)*x3(1,1)
    +3*x1(1,3)*x2(1,3)*x3(0,3)
    +1*x1(1,1)*x2(0,3)*x3(0,0)
    +1*x1(0,3)*x2(0,3)*x3(0,2)
    +1*x1(1,1)*x2(0,3)*x3(0,1);
# D&M: 8/7
4:3:
+1*x1(1,1)*x2(0,1)*x3(2,3)
    +2*x1(1,3)*x2(2,2)*x3(0,3)
    +3*x1(0,2)*x2(0,1)*x3(1,1)
    +2*x1(0,2)*x2(0,0)*x3(0,3)
    +3*x1(3,3)*x2(0,2)*x3(0,1)
    +3*x1(3,3)*x2(1,1)*x3(2,2)
    +2*x1(2,3)*x2(1,1)*x3(1,3);
# P&A: 8/7
4:3:
+2*x1(0,3)*x2(0,3)*x3(1,1)
    +1*x1(3,3)*x2(2,3)*x3(1,3)
    +3*x1(1,2)*x2(0,1)*x3(2,2)
    +3*x1(1,1)*x2(3,3)*x3(2,2)
    +3*x1(2,2)*x2(3,3)*x3(0,3)
    +2*x1(0,1)*x2(1,3)*x3(1,1);

# P&A: 8/7
4:3:
    +3*x1(0,2)*x2(3,3)*x3(3,3)
    +1*x1(1,1)*x2(0,3)*x3(0,3)
    +2*x1(1,1)*x2(1,1)*x3(1,2)
    +1*x1(0,1)*x2(1,1)*x3(3,3)
    +2*x1(0,1)*x2(2,3)*x3(1,1)
    +2*x1(0,1)*x2(0,1)*x3(0,0)
    +3*x1(2,3)*x2(1,3)*x3(0,2);

(data file; tt7.out)
```

APPENDIX B

Program Listing

THE FOLLOWING ARE THE MODULES USED FOR THE MVL-CAD. THE ENTIRE PROGRAM IS ABOUT 4,000 LINES LONG.
HERE IS A BRIEF OVERVIEW OF WHAT IS INCLUDED.

(1) VERSION BANNER, HELP DISPLAY AND GLOBAL DATA STRUCTURES FOR MVL-CAD.

(2) main.c

THIS MODULE 1) PROCESSES COMMAND LINE ARGUMENTS 2) OPENS FILES AND 3) CONTROLS THE PARSING, LOGIC MINIMIZATION AND OUTPUT FUNCTIONS. ALSO, IT INCLUDES Gold(), WHICH IMPLEMENTS THE GOLD HEURISTIC (WHICH CHOOSES BETWEEN POMPER-ARMSTRONG AND DUECK-MILLER,PREFERRING DUECK-MILLER ON TIES.)

(3) defs.c

THIS MODULE INCLUDES ALL GLOBAL VARIABLE DEFINITIONS.

(4) parse.l

THIS MODULE IS A LEX SPECIFICATION FILE FOR THE INPUT SCANNER/PARSER FOR THE MVL-CAD. IT SPECIFIES THE GRAMMAR OF EXPRESSIONS INPUT BY THE USER. ALSO, IT INCLUDES verify(), WHICH ALLOWS THE USER TO DETERMINE THE FUNCTION VALUE STORED.

(5) dm.c

THIS MODULE IMPLEMENTS THE DUECK-MILLER HEURISTIC FOR MINIMIZING AN EXPRESSION.
THE FOLLOWING SUBROUTINS ARE INCLUDED IN THIS MODULE.

a) Dueck_Miller()
   - performs the Dueck and Miller heuristic on the input expression and returns the number of implicants used to cover it.

b) _cf(E,X)
   - computes the clustering factors at X, where X is a vector of

coordinates of a non-zero minterm which has the smallest value.

- returns the clustering factor, an integer.

c) *mim(E)

- finds the most isolated minterm determined by the minterm with smallest CF and returns a vector of integers representing the coordinate of it or NULL if no more minterms
- the function value at that location is also returned as the last integer in the vector.

d) valid_implicant(I)

- checks implicant validity and returns 1 if the proposed implicant is valid; else returns 0.(An implicant is valid if it is indeed an implicant of the function.)

e) compute_rbc(I)

- accepts implicant I and calculates the relative break count of it.

f) *pick_implicant(X)

- finds the most isolated minterm according to the smallest CF.
- generates all implicants that covers this minterm.
- computes the rbc for each and chooses the implicant with the smallest rbc (best implicant).
- returns the best implicant.

(6) pa.c

THIS MODULE IMPLEMENTS THE POMPER-ARMSTRONG HEURISTIC FOR MINIMIZING AN EXPRESSION.

THE FOLLOWING SUBROUTINES ARE INCLUDED IN THIS MODULE.

a) Pomper_Armstrong()

- performs the Pomper-Armstrong heuristic on the input expression and returns the number of implicants used to cover it.

b) *mim(E)

- finds the most isolated minterm in the input expression based on the random weight generated by random number generator.
- returns a vector of integers representing the coordinates of the most isolated minterm or NULL if no more minterms.

- the value of the function corresponding to the most isolated minterm is also returned as the last integer in the vector.

c) random()

- returns a random number using a linear congruential method.

d) valid_implicant(I)

- checks implicant validity and if valid, returns 1 and the num_zeroed, the number of minterms driven to zero by this implicant when subtracted, else returns 0.

e) *pick_implicant(X)

- finds the most isolated minterm randomly and then produces all implicants that cover this minterm.
- computes num_covered, the number of minterms covered by this implicant.
- picks the best implicant which drives the maximum number of minterms to zero. If there are more than one such implicant the largest implicant is chosen.
- returns the best implicant.

(7) common.c

THIS MODULE INCLUDES HEURISTIC SUPPORT FUNCTIONS COMMON TO DUECK-MILLER AND POMPER-ARMSTRONG.

THE FOLLOWING SUBROUTINES ARE INCLUDED IN THIS MODULE.

a) *_eval(E,X)

- evaluates the expression at (x1,x2,x3,....,xn)

b) *next_coord(T)

- computes the next possible coordinate for term.
- returns an integer vector(X) containing the coordinates.

c) *gen_bounds(X)

- generates the permissible bounds around location X in the working expression for use in picking the next implicant that covers X.
- returns a bounds array.

d) *next_implicant(B)

- chooses X as the first implicant.
- On each call, returns the next implicant with a bound array(B) from *gen_bounds() that specifies the implicant.

e) subtract_implicant(I)

- adds the best implicant(I) to the working expression as a negative term (negated coefficient).
- Also adds the best implicant to the final expression.

(8) alloc.c

THIS MODULE INCLUDES ALL MEMORY ALLOCATION FUNCTIONS, SUCH AS alloc_term(), alloc_bound(),and dealloc(),etc. IT IS USED TO ALLOCATE SPACE FOR A TERM ARRAY, BOUND ENTRIES AND DEALLOCATE OR DUPULICATE THE EXPRESSION.

(9) mvltest.c

THIS MODULE IS A TEST SOURCE FILE GENERATOR. IT GENERATES ANY TYPE OF MVL EXPRESSIONS ACCORDING TO THE SPECIFICATION GIVEN BY THE USER.

(10) Makefile

THIS MODULE KEEPS TRACK OF WHICH SOURCE FILES ARE DEPENDENT ON OTHERS AND PROVIDES FOR AN EFFICIENT RECOMPILATION OF A PROGRAM.

```
/* mvl

    - A multi-valued logic expression compiler/analyzer/optimizer

    LCDR John M. Yurchak
    LT Hoon Seop Lee
    Dr. Jon T. Butler

    Copyright 1988  U.S. Naval Postgraduate School, Monterey, California
    All rights reserved                                                  */



#include "defs.h"
#include <ctype.h>

/* Version banner and help display */

static char
*version = "\nMVL  V1.0  Copyright 1988  Naval Postgraduate School\n\n",
*usage = "usage: mvl [-eimrsvqDPG] [-0.0] [-oFile] [-xFile] [file ...]\n",
*help[] = {
    "    -e         - Print the original expression, as parsed",
    "    -i         - Print each implicant found",
    "    -m         - Print the Karnaugh map of the original and each",
    "                 successive expression",
    "                 WARNING: impractical for nvar > 3",
    "    -v         - Verify the minimal solution",
    "    -s         - Print statistics on selected heuristics",
    "    -q         - Quiet, don't print final results",
    "    -r         - Permit negative coefficients in terms",
    "    -0.0       - Build source expressions for all solutions",
    "                 whose ratio to the original number of terms",
    "                 exceeds this number and output them to a file",
    "                 (default is \"x.mvl\")",
    "    -D         - Execute the Dueck & Miller heuristic (default)",
    "    -P         - Execute the Pomper & Armstrong heuristic",
    "    -G         - Execute the Gold heuristic",
    "    -xFile     - Output source expressions built using -0.0 to \"File\"",
    "                 instead of \"x.mvl\"",
    "    -oFile     - Ouput formatted solutions to \"File\"",
    NULL
};
```

```
/* Global data structures ------------------------------------------------ */

    /* Logic expressions:

        E_orig
            - holds the original input expression as parsed

        E_work
            - a copy a E_orig
            - implicants are subtracted from this expression as terms
            during the coures of optimization

        E_final[]
            - the result expression (starts out empty)
            - each term is one implicant found during optimization
            - each heuristic has its own E_final (for comparison)
    */

Expression
    E_orig = { 0,0,0,NULL },
    E_work = { 0,0,0,NULL },
    E_final[2] = {
        { 0,0,0,NULL },
        { 0,0,0,NULL }
    };

int HEUR;                               /* Current heuristic
                                         * HEUR indexes into E_final[]
                                         * depending upon the currently
                                         * active heuristic
                                         */
int FINAL;                              /* Index of the selected final
                                         * expression
                                         */

char
    expression[MAX_BUF+1],              /* Parser working buffer */
    if_name[MAX_PATH+1],                /* Input file name */
    xf_name[MAX_PATH+1] = "x.mv1\0",    /* Expression output file name */
    *tmp_name = "mv1XXXXXX",
    of_name[MAX_PATH+1];                /* Output file name (for result) */

int token_type;                         /* Scanner token code */
int completed;                          /* Flag:
                                         *  1 if a complete sentence has been
                                         *  recognized, 0 otherwise
                                         */
jmp_buf eof_context;                    /* Saved context for lex end of file */

/* user options */

int r_flag = 0,
    e_flag = 0,
    i_flag = 0,
    m_flag = 0,
    s_flag = 0,
    v_flag = 0,
    D_flag = 1,
    F_flag = 0,
    G_flag = 0,
    q_flag = 0;

int nvar,           /* Number of variables in the current expression */
    radix;          /* Radix of the current expression */
```

```
int tot_expr = 0,    /* Total number of expressions optimized */
    tot_terms = 0;   /* Total number of terms in all input expressions */

MVL_stats
    *STAT,           /* Pointer to the current statistics record */
    DM_stat,         /* Stats record for the D&M heuristic */
    PA_stat;         /* Stats record for the P&A heuristic */

float   FO_ratio = 0.0;

FILE    *xfile;

/*
    - yywrap() is called by lex on EOF.
    - Thus, we execute a non-local goto back to the setjmp call which
    saved the context, at which point we open the next input file.
    - Notice that yywrap() never actually returns.
*/
yywrap()
{
    longjmp(eof_context,1);
}
```

```
main(argc,argv)
char    *argv[];
/* ---------------------------------------------------------------------------
    :function:
        - Process command line arguments
        - Open files
        - Control the parsing, optimization and output functions
--------------------------------------------------------------------------- */
{
    register ap;
    char    *p;

    fprintf(stderr,"%s",version);

    /* Do the options */

    for (ap=1; ap < argc; ap++) {
        p = argv[ap];
        if (*p++ == '-') {
            /* loop through an option string */
            while (*p) {
                switch (*p++) {
                case 'D':
                    D_flag++;
                    break;
                case 'P':
                    P_flag++;
                    if (D_flag==1)
                        D_flag--;
                    break;
                case 'G':
                    G_flag++;
                    D_flag++;
                    P_flag++;
                    break;
                case 'v':
                    v_flag++;
                    break;
                case 'x':
                    if (*p) {
                        strcpy(xf_name,p);
                        /* terminate the while loop */
                        *p = '\0';
                    }
                    else {
                        fprintf(stderr,"mvl: filename expected after -x\n");
                        fprintf(stderr,"%s",usage);
                        exit(1);
                    }
                    break;
                case 'o':
                    if (*p) {
                        strcpy(of_name,p);
                        /* terminate the while loop */
                        *p = '\0';
                    }
                    else {
                        fprintf(stderr,"mvl: filename expected after -o\n");
                        fprintf(stderr,"%s",usage);
                        exit(1);
                    }
                    break;
                case 'q':
                    q_flag++;
                    break;
```

```
            case 's':
                s_flag++;
                break;
            case 'r':
                r_flag++;
                break;
            case 'm':
                m_flag++;
                break;
            case 'e':
                e_flag++;
                break;
            case 'i':
                i_flag++;
                break;
            case '.':
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                p--;
                sscanf(p,"%f",&FO_ratio);
                *p = '\0';
                break;
            case '-':
                fprintf(stderr,"%s",usage);
                for (ap=0; help[ap]; ap++)
                    fprintf(stderr,"%s\n",help[ap]);
                exit(0);
            default:
                fprintf(stderr,"mvl: unknown switch - %c\n",p[-1]);
                fprintf(stderr,"%s",usage);
                exit(1);
            }
        }
    }
}

/* Erase the old expression file (if there is one) */
unlink(xf_name);

/* If the user specified an output file ... */
if (of_name[0]) {
    /* erase the existing output file and open the new one (append) */
    unlink(of_name);
    mktemp(tmp_name);
    if ((yyout=fopen(tmp_name,"a")) == NULL) {
        fprintf(stderr,"mvl: can't open temp file - %s\n",tmp_name);
        exit(1);
    }
}

/* Init the scanner */
token_type = TOK_EOL;

/* For each input file */
for (ap=1; ap < argc; ap++) {
    p = argv[ap];
    /* skip the option arguments */
    if (*p == '-')
```

```
            continue;
        strcpy(if_name,p);
        if ((yyin=fopen(if_name,"r")) == NULL) {
            fprintf(stderr,"mv1: can't open %s\n",if_name);
            exit(1);
        }
        init_stats(&DM_stat);
        init_stats(&PA_stat);

        /* Set the EOF context */
        if (setjmp(eof_context) == 0) {

            /* While still expressions to parse in this file ... */
            while (expr() > 0) {
                nvar = E_orig.nvar;
                radix = E_orig.radix;
                tot_terms += E_orig.nterm;
                tot_expr++;

                if (D_flag) { /* do the Dueck & Miller */
                    Dueck_Miller();
                    FINAL = D_M;
                }
                if (P_flag) { /* do the Pomper & Armstrong */
                    Pomper_Armstrong();
                    if (!D_flag)
                        FINAL = P_A;
                }
                if (G_flag) { /* do the Gold (which chooses between above) */
                    Gold();
                    if (of_name[0]) /* output the final expression */
                        output_expr(HEUR);
                    FINAL = HEUR;
                }
                else {
                    if (of_name[0]) {
                        if (D_flag) /* output the final expression */
                            output_expr(D_M);
                        else if (P_flag)
                            output_expr(P_A);
                    }
                }
                verify();
                dealloc_expr(&E_orig);
            }
        }
        else if (!completed)
            syntax_error(ERR_UNEXP_EOF);

        /* Print the statistics */
        if (s_flag) {
            if (D_flag)
                print_DM_stats();
            if (P_flag)
                print_PA_stats();
        }
        fclose(yyin);
    }
    if (of_name[0]) {

        /* Build the output file for the layout generator by appending
         * the list of final expressions in the temporary file to
         * a header consisting of radix, inputs and outputs.
         *
         * Final format is:
         *
```

```
         *   radix  inputs(nvar)  outputs(tot_expr)
         *   coeff
         *   lower upper
         *   lower upper
         *   ...
         *   ...
         *   99 (sentinel)
         *   coeff
         *   lower upper
         *   lower upper
         *   ...
         *   ...
         *   99 (sentinel)
         *   ...
         *   etc
         */

        /* Close the temporary file, reopen for reading and open
         * the final output.
         */
        fclose(yyout);
        if ((yyin = fopen(tmp_name,"r")) == NULL) {
            fprintf(stderr,"mvl: can't open temp file - %s\n",tmp_name);
            exit(1);
        }
        if ((yyout = fopen(of_name,"w")) == NULL) {
            fprintf(stderr,"mvl: can't open output file - %s\n",of_name);
            exit(1);
        }

        /* Build and output the header */
        sprintf(expression,"%d %d %d\n",radix,nvar,tot_expr);
        fputs(expression,yyout);

        /* Append the temp file, then delete it */
        while (fgets(expression,MAX_BUF,yyin) != NULL)
            fputs(expression,yyout);
        fclose(yyout);
        fclose(yyin);
        unlink(tmp_name);
    }
}
```

```
Gold()
/* ---------------------------------------------------------------------------
    :function:
        - Implement the Gold heuristic (which chooses between P&A and D&M,
        preferring D&M on ties).
    :globals:
        E_final[]
        E_orig
        HEUR
    :called_by:
        main()
--------------------------------------------------------------------------- */
{
    char    *p;

    if (E_final[D_M].nterm < E_final[P_A].nterm) {
        HEUR = D_M;
        p = "D&M";
    }
    else if (E_final[D_M].nterm > E_final[P_A].nterm) {
        HEUR = P_A;
        p = "P&A";
    }
    else  {
        HEUR = D_M;
        p = "===";
    }
    if (!q_flag) {
        printf("  Gold(%s): %4d/%-4d implicants - %4.2f%%\n\n",
            p,E_final[HEUR].nterm,E_orig.nterm,
            (float)E_final[HEUR].nterm/(float)E_orig.nterm);
    }
}
```

```
print_map()
/* ---------------------------------------------------------------------------
    :function:
        - Print the Karnaugh map of E_work in its present state
    :globals:
        E_work
    :called_by:
        Dueck_Miller()
        Pomper_Armstrong()
    :calls:
        _eval()
--------------------------------------------------------------------------- */
{
    register    i,j;
    int X[MAX_VAR+1];
    int *V;

    for (i=0; i < nvar; i++) X[i] = 0;
    for (i=0; i < nvar;) {
        V = _eval(&E_work,X);
        printf("%3d%c",V[EVAL],V[HLV]?'.':' ');
        X[i]++;
        for (;i < nvar;) {
            if (X[i] >= radix) {
                X[i] = 0;
                if (i < 2)
                    printf("\n");
                i++;
                X[i]++;
            }
            else {
                i = 0;
                break;
            }
        }
    }
}
```

```
print_implicant(s,X,I)
char    *s;
int     *X;
Term    *I;
/* ---------------------------------------------------------------------------
    :function:
        - Print the Most Isolated Minterm X and the implicant selected
        to cover it I.
    :called_by:
        Dueck_Miller()
        Pomper_Armstrong()
--------------------------------------------------------------------------- */
{
    register i;

    printf("  %s MIM: %2d",s,X[nvar]);
    for (i=0; i < nvar; i++)
        printf("*X%d(%2d,%2d)",i+1,X[i],X[i]);
    printf("\n");

    printf("      Imp: %2d",I->coeff);
    for (i=0; i < nvar; i++)
        printf("*X%d(%2d,%2d)",i+1,I->B[i].lower,I->B[i].upper);
    printf("\n\n");
}

print_terms(E)
Expression *E;
/* ---------------------------------------------------------------------------
    :function:
        - Print the expression pointer to by E
    :called_by:
        Dueck_Miller()
        Pomper_Armstrong()
--------------------------------------------------------------------------- */
{
    int i,j;

    printf("\nExpression:\n\n");
    printf("  radix:  %2d\n",E->radix);
    printf("  nvars:  %2d\n",E->nvar);
    printf("  nterms: %2d\n",E->nterm);
    for (i=0; i < E->nterm; i++) {
        printf("  coeff: %d\n",E->T[i].coeff);
        for (j=0; j < E->nvar; j++)
            printf("    X%d(%d,%d)\n",
                j+1,E->T[i].B[j].lower,E->T[i].B[j].upper);
    }
    printf("\n");
}
```

```
output_expr(heur)
int heur;
/* ---------------------------------------------------------------------------
    :function:
        - Append the final expression to the temporary file.
        - heur indexes into E_final.
        - If the final expression is not at least as good as the original
        input expression, output the original.
        - Output format as follows:

            coeff
            lower upper
            lower upper
            ...
            99  (sentinel)
    :globals:
        yyout
        stdout
        E_orig
        E_final
    :called_by:
        main()
--------------------------------------------------------------------------- */
{
    register i,j;
    Expression  *E;

    if (yyout != stdout) {
        if (E_orig.nterm < E_final[heur].nterm)
            E = &E_orig;
        else
            E = &E_final[heur];
        for (i=0; i < E->nterm; i++) {
            fprintf(yyout,"%d\n",E->T[i].coeff);
            for (j=0; j < E->nvar; j++)
                fprintf(yyout,"%d %d\n",
                    E->T[i].B[j].lower,
                    E->T[i].B[j].upper);
        }
        fprintf(yyout,"99\n");
    }
}
```

```
print_source(E,s,n)
Expression  *E;
char    *s;
/* ---------------------------------------------------------------------------
    :function:
        - Print the expression pointed to by E to the file xf_name in
        the form recognizeable by the parser.
    :globals:
        xf_name
    :called_by:
        Dueck_Miller()
        Pomper_Armstrong()
--------------------------------------------------------------------------- */
{
    register i,j;
    FILE    *f;

    if ((f = fopen(xf_name,"a")) == NULL) {
        fprintf(stderr,"mvl: can't open %s\n",xf_name);
        return;
    }

    fprintf(f,"# %s: %d/%d\n",s,n,E->nterm);
    fprintf(f,"%d:%d:",E->radix,E->nvar);
    for (i=0; i < E->nterm; i++) {
        fprintf(f,"\n    +%d",E->T[i].coeff);
        for (j=0; j < E->nvar; j++) {
            fprintf(f,"*x%d(%d,%d)",
                j+1,E->T[i].B[j].lower,E->T[i].B[j].upper);
        }
    }
    fprintf(f,";\n");
    fclose(f);
}
```

```
init_stats(S)
MVL_stats   *S;
/* ---------------------------------------------------------------------------
    :function:
        - Initialize statistics structure pointer to by S
    :called_by:
        main()
--------------------------------------------------------------------------- */
{
    tot_expr = 0;
    tot_terms = 0;
    S->calls_eval = 0;
    S->calls_cf = 0;
    S->calls_mim = 0;
    S->calls_gen_bounds = 0;
    S->calls_pick_implicant = 0;
    S->calls_next_implicant = 0;
    S->calls_valid_implicant = 0;
    S->calls_compute_rbc = 0;
}

print_PA_stats()
/* ---------------------------------------------------------------------------
    :function:
        - Print the stats on the P&A heuristic
    :globals:
        tot_terms
        tot_expr
        radix
        nvar
        PA_stat
    :called_by:
        main()
--------------------------------------------------------------------------- */
{
    printf("Statistics for P&A:\n\n");

    printf("       Num     Num      Num       Avg Num\n");
    printf("Radix  Var     Expr     Terms    Term/Expr\n");
    printf("%4d  %4d  %6d  %6d  %8.2f\n\n",
        radix,nvar,tot_expr,tot_terms,(float)tot_terms/(float)tot_expr);
    printf("                  Eval      Pick      Gen       Next      Valid\n");
    printf("                  Expr      MIM      Bounds     Impl      Impl\n");
    printf("Tot:     %11d %8d %8d %8d %8d\n",
        PA_stat.calls_eval,PA_stat.calls_mim,
        PA_stat.calls_gen_bounds,PA_stat.calls_next_implicant,
        PA_stat.calls_valid_implicant);
    printf("Avg/Expr:  %11.2f %8.2f %8.2f %8.2f %8.2f\n",
        (float)PA_stat.calls_eval/(float)tot_expr,
        (float)PA_stat.calls_mim/(float)tot_expr,
        (float)PA_stat.calls_gen_bounds/(float)tot_expr,
        (float)PA_stat.calls_next_implicant/(float)tot_expr,
        (float)PA_stat.calls_valid_implicant/(float)tot_expr);
    printf("Avg/Term:  %11.2f %8.2f %8.2f %8.2f %8.2f\n\n",
        (float)PA_stat.calls_eval/(float)tot_terms,
        (float)PA_stat.calls_mim/(float)tot_terms,
        (float)PA_stat.calls_gen_bounds/(float)tot_terms,
        (float)PA_stat.calls_next_implicant/(float)tot_terms,
        (float)PA_stat.calls_valid_implicant/(float)tot_terms);
}
```

```
print_DM_stats()
/* ---------------------------------------------------------------------------
    :function:
        - Print the stats on the D&M heuristic
    :globals:
        tot_terms
        tot_expr
        radix
        nvar
        DM_stat
    :called_by:
        main()
--------------------------------------------------------------------------- */
{
    printf("Statistics for D&M:\n\n");

    printf("       Num     Num      Num       Avg Num\n");
    printf("Radix  Var     Expr     Terms    Term/Expr\n");
    printf("%4d  %4d  %6d  %6d  %8.2f\n\n",
        radix,nvar,tot_expr,tot_terms,(float)tot_terms/(float)tot_expr);
    printf("                 Eval      Comp      Pick      Comp      Gen       Next      Valid\n
    printf("                  Expr       CF       MIM       RBC      Bounds    Impl      Impl\n"
    printf("Tot:     %11d %8d %8d %8d %8d %8d %8d\n",
        DM_stat.calls_eval,DM_stat.calls_cf,DM_stat.calls_mim,
        DM_stat.calls_compute_rbc,
        DM_stat.calls_gen_bounds,DM_stat.calls_next_implicant,
        DM_stat.calls_valid_implicant);
    printf("Avg/Expr:  %11.2f %8.2f %8.2f %8.2f %8.2f %8.2f %8.2f\n",
        (float)DM_stat.calls_eval/(float)tot_expr,
        (float)DM_stat.calls_cf/(float)tot_expr,
        (float)DM_stat.calls_mim/(float)tot_expr,
        (float)DM_stat.calls_compute_rbc/(float)tot_expr,
        (float)DM_stat.calls_gen_bounds/(float)tot_expr,
        (float)DM_stat.calls_next_implicant/(float)tot_expr,
        (float)DM_stat.calls_valid_implicant/(float)tot_expr);
    printf("Avg/Term:  %11.2f %8.2f %8.2f %8.2f %8.2f %8.2f %8.2f\n\n",
        (float)DM_stat.calls_eval/(float)tot_terms,
        (float)DM_stat.calls_cf/(float)tot_terms,
        (float)DM_stat.calls_mim/(float)tot_terms,
        (float)DM_stat.calls_compute_rbc/(float)tot_terms,
        (float)DM_stat.calls_gen_bounds/(float)tot_terms,
        (float)DM_stat.calls_next_implicant/(float)tot_terms,
        (float)DM_stat.calls_valid_implicant/(float)tot_terms);
}


fatal(s)
char    *s;
{
    fprintf(stderr,"%s\n",s);
    exit(1);
}
```

```
/* defs.h

    - External definitions for mvl

*/

#include <stdio.h>
#include <setjmp.h>

#define MAX_BUF 4096
#define MAX_PATH 64
#define MAX_VAR 32
#define MAX_INT 32765

#define YES 1
#define NO  0

#define EVAL    0
#define HLV     1

#define D_M     0
#define P_A     1

/* parser definitions and data structures ---------------------------------- */

#define TOK_EOF     0
#define TOK_NUMBER  2
#define TOK_VAR     3
#define TOK_COLON   4
#define TOK_STAR    5
#define TOK_LPAREN  6
#define TOK_COMMA   7
#define TOK_RPAREN  8
#define TOK_SIGNED  9
#define TOK_DONTCARE 10
#define TOK_SEMI    11
#define TOK_PLUS    12
#define TOK_MINUS   13

#define TOK_EOL     14

#define TOK_SYNERR  -1

#define ERR_RADIX_RNG   -2
#define ERR_NVAR_RNG    -3
#define ERR_COEFF_RNG   -4
#define ERR_VARINDEX_RNG    -5
#define ERR_DUP_VAR     -6
#define ERR_LOWER_RNG   -7
#define ERR_UPPER_RNG   -8
#define ERR_UPPER_LOWER -9
#define ERR_UNEXP_EOF   -10

extern char expression[MAX_BUF+1];
extern int token_type;

extern FILE *yyin,*yyout;

/* MVL definitions and data structures */

typedef struct expr_struct Expression;
typedef struct term_struct Term;
typedef struct bound_struct Bound;
typedef struct term_struct Implicant;

struct expr_struct {
```

```
    int radix,
        nvar,
        nterm;
    Term    *T;
);

struct term_struct {
    int coeff;
    Bound   *B;
);

struct bound_struct {
    int lower,
        upper;
);

extern Expression
    E_orig,
    E_work,
    E_final[2];

extern int  HEUR,FINAL;

extern  int
    nvar,
    radix;

Term *alloc_term();
Bound *alloc_bound();

extern char
    if_name[MAX_PATH+1],
    xf_name[MAX_PATH+1],
    of_name[MAX_PATH+1];

extern  r_flag,
        e_flag,
        i_flag,
        x_flag,
        m_flag,
        G_flag,
        v_flag,
        q_flag;

extern FILE *xfile;
extern int completed;

extern
int tot_expr,
    tot_terms;

struct mvl_stats {
    long
        calls_eval,
        calls_cf,
        calls_mim,
        calls_gen_bounds,
        calls_next_implicant,
        calls_pick_implicant,
        calls_valid_implicant,
        calls_compute_rbc;
);

typedef struct mvl_stats MVL_stats;

extern MVL_stats
```

```
    *STAT,
    DM_stat,
    PA_stat,
    G_stat;

extern float    FO_ratio;

int *mim();
int *next_coord();
Bound *gen_bounds();
Term *next_implicant();
Term *pick_implicant();
int *eval(),*_eval();
```

```
/* parse.l

    - lex specification file for the input scanner/parser for the MVL
    parser/compiler

expression grammar:

    expr :=
            radix ':' num_var ':' term_list

    term_list :=
            term ';'
        |   term '+' term_list

    term :=
            coeff '*' var_list

    var_list :=
            variable
        |   variable '*' var_list

    variable :=
            var_id '(' limit ',' limit ')'

verivication test grammar:

    verification :=
            variable ';'
        |   variable verification

constraints:

    radix >= 2
    num_var >= 1
    1 <= coeff <= radix (radix == don't care)
    0 <= limit <= radix-1  and  lower_limit <= upper_limit
    "x1" <= var_id <= "x(num_var)"

data structures

    expression

        [ radix ]
        [ nvar ]
        [ nterm ]
        [ term * ]

        - An expression structure contains the radix of the logic
```

```
        expression, the number of variables, the number of variables
        and a pointer to an array of terms

    term

        [ coeff ]
        [ bounds *]

        - A term structure contains a coefficient and a pointer to a
        list of bounds (length equal to the number of variables
        allowed in the expression)

    bounds

        [ lower ][ upper ]
        [ lower ][ upper ]
            .
            .
            .

        - A bounds list is an array of pairs (lower and upper bounds)
        which may range over the radix of the expression, one pair
        for each possible variable in the term

*/

#include "defs.h"

%}
%%
"#"[^\n]*\n {
}

[\t ]* {
}

[0-9]+ {
    strcat(expression,yytext);
    return(TOK_NUMBER);
}

[+-][0-9]+ {
    strcat(expression,yytext);
    return(TOK_SIGNED);
}

"?" {
    strcat(expression,yytext);
    return(TOK_DONTCARE);
}

[xX][1-9][0-9]* {
    strcat(expression,yytext);
    return(TOK_VAR);
}

":" {
    strcat(expression,yytext);
    return(TOK_COLON);
}

"*" {
    strcat(expression,yytext);
    return(TOK_STAR);
}
```

```
"(" {
    strcat(expression,yytext);
    return(TOK_LPAREN);
}

"+" {
    strcat(expression,yytext);
    return(TOK_PLUS);
}

"-" {
    strcat(expression,yytext);
    return(TOK_MINUS);
}

"," {
    strcat(expression,yytext);
    return(TOK_COMMA);
}

")" {
    strcat(expression,yytext);
    return(TOK_RPAREN);
}

. {
    return(NO);
}

";" {
    strcat(expression,yytext);
    return(TOK_SEMI);
}

[ \t\f\n\r] {
    expression[0] = '\0';
}
%%

/* parser NT functions for expressions */

expr()
{
    int rad,nvar;

    init_expr();
    expression[0] = '\0';
    if (!yytext[0] || match(";"))
        next_token();
    completed = 0;
    if (token_type == TOK_EOF)
        return(0);

    if (!radix_val()) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    sscanf(yytext,"%d",&rad);
    if (rad < 2) {
        syntax_error(ERR_RADIX_RNG);
        return(NO);
    }
    E_orig.radix = rad;

    next_token();
    if (!match(":")) {
```

```
            syntax_error(TOK_SYNERR);
            return(NO);
        )

        next_token();
        if (!num_var()) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )

        sscanf(yytext,"%d",&nvar);
        if (nvar < 1) {
            syntax_error(ERR_NVAR_RNG);
            return(NO);
        )
        E_orig.nvar = nvar;

        next_token();
        if (!match(":")) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )

        next_token();
        if (!term_list()) {
            syntax_erro·(TOK_SYNERR);
            return(NO);
        )

        if (!match(";")) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )
        completed = 1;
        return(YES);
    )

    radix_val()
    {
        if (token_type == TOK_NUMBER)
            return(YES);
        return(NO);
    )

    num_var()
    {
        if (token_type == TOK_NUMBER)
            return(YES);
        return(NO);
    )

    term_list()
    {
        register i;
        if (!term())
            return(NO);
        if (match("+") | match("-")) {
            next_token();
            while (term_list());
        )
        else
            while (term_list());
        for (i=0; i < E_orig.nvar; i++) {
            if (E_orig.T[E_orig.nterm-1].B[i].lower == -1)
                E_orig.T[E_orig.nterm-1].B[i].lower = 0;
        )
```

```
    return(YES);
}

term()
{
    int coeff;
    if ((token_type == TOK_NUMBER) || (token_type == TOK_SIGNED)) {
        sscanf(yytext,"%d",&coeff);
        if (!r_flag) {
            if ((coeff < 1) || (coeff >= E_orig.radix)) {
                syntax_error(ERR_COEFF_RNG);
                return(NO);
            }
        }
    }
    else if (token_type == TOK_DONTCARE) {
        coeff = E_orig.radix;
    }
    else {
        return(NO);
    }

    E_orig.nterm++;
    E_orig.T = alloc_term(E_orig.T,coeff,E_orig.nterm);

    next_token();
    if (!match("*")) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    next_token();
    if (!var_list())
        return(NO);
    return(YES);
}

var_list()
{
    if (!variable())
        return(NO);
    next_token();
    if (match("*")) {
        next_token();
        while (var_list());
    }
    return(YES);
}

variable()
{
    int varindex,lower,upper;
    if (!var_id())
        return(NO);
    sscanf(&yytext[1],"%d",&varindex);
    if ((varindex < 1) || (varindex > E_orig.nvar)) {
        syntax_error(ERR_VARINDEX_RNG);
        return(NO);
    }
    varindex--;
    if (E_orig.T[E_orig.nterm-1].B[varindex].lower != -1) {
        syntax_error(ERR_DUP_VAR);
        return(NO);
    }

    next_token();
    if (!match("(")) {
```

```
            syntax_error(TOK_SYNERR);
            return(NO);
        )
        next_token();
        if (!limit()) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )
        sscanf(yytext,"%d",&lower);
        if ((lower < 0) )| (lower > E_orig.radix-1)) {
            syntax_error(ERR_LOWER_RNG);
            return(NO);
        )

        next_token();
        if (!match(",")) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )
        next_token();
        if (!limit()) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )
        sscanf(yytext,"%d",&upper);
        if ((upper < 0) )| (upper > E_orig.radix-1)) {
            syntax_error(ERR_UPPER_RNG);
            return(NO);
        )
        if (upper < lower) {
            syntax_error(ERR_UPPER_LOWER);
            return(NO);
        )

        next_token();
        if (!match(")")) {
            syntax_error(TOK_SYNERR);
            return(NO);
        )

        E_orig.T[E_orig.nterm-1].B[varindex].lower = lower;
        E_orig.T[E_orig.nterm-1].B[varindex].upper = upper;
        return(YES);
    )

    var_id()
    {
        if (token_type == TOK_VAR)
            return(YES);
        return(NO);
    )

    limit()
    {
        if (token_type == TOK_NUMBER)
            return(YES);
        return(NO);
    )

    /* verification NT functions */

    verify()
    {
        register i;
        Bound   B[MAX_VAR+1];
        Term    T;
```

```
    int first,first_time = 1;
    int *value,*X;

    for (;;) {
        if (match(";"))
            next_token();
        T.B = B;
        for (i=0; i < nvar; i++)
            B[i].lower = -1;
        if (!ver_variable(B))
            return;
        completed = 0;
        while (ver_variable(B));
        if (!match(";"))
            syntax_error(TOK_SYNERR);
        completed = 1;
        if (!v_flag)
            continue;

        first = 1;
        if (first_time) {
            printf("Verification:\n\n");
            first_time = 0;
        }
        for (i=0; i < nvar; i++)
            printf("  X%d(%d,%d)\n",i+1,B[i].lower,B[i].upper);
        printf("\n");
        while ((X = next_coord(&T,first)) != NULL) {
            first = 0;
            value = _eval(&E_final[FINAL],X);
            printf("  (");
            for (i=0; i < nvar; i++) {
                printf("%d",X[i]);
                if (i+1 < nvar) printf(",");
            }
            printf(") = %3d\n",value[EVAL]);
        }
        printf("\n");
    }
}

ver_variable(B)
Bound   *B;
{
    int varindex,lower,upper;
    if (!var_id())
        return(NO);
    sscanf(&yytext[1],"%d",&varindex);
    if ((varindex < 1) || (varindex > nvar)) {
        syntax_error(ERR_VARINDEX_RNG);
        return(NO);
    }
    varindex--;
    if (B[varindex].lower != -1) {
        syntax_error(ERR_DUP_VAR);
        return(NO);
    }

    next_token();
    if (!match("(")) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    next_token();
    if (!limit()) {
        syntax_error(TOK_SYNERR);
```

```
        return(NO);
    }
    sscanf(yytext,"%d",&lower);
    if ((lower < 0) || (lower > radix-1)) {
        syntax_error(ERR_LOWER_RNG);
        return(NO);
    }

    next_token();
    if (!match(",")) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    next_token();
    if (!limit()) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    sscanf(yytext,"%d",&upper);
    if ((upper < 0) || (upper > radix-1)) {
        syntax_error(ERR_UPPER_RNG);
        return(NO);
    }
    if (upper < lower) {
        syntax_error(ERR_UPPER_LOWER);
        return(NO);
    }

    next_token();
    if (!match(")")) {
        syntax_error(TOK_SYNERR);
        return(NO);
    }
    B[varindex].lower = lower;
    B[varindex].upper = upper;
    next_token();
    return(YES);
}

/* scanner utility functions ------------------------------------------- */

syntax_error(code)
{
    static char *errors[] = {

                                "",
        /* TOK_SYNERR       */  "syntax error",

        /* ERR_RADIX_RNG    */  "radix out of range (>= 2)",
        /* ERR_NVAR_RNG     */  "# of var out of range (>= 1)",
        /* ERR_COEFF_RNG    */  "coeff out of range (1 <= coeff <= radix-1)",
        /* ERR_VARINDEX_RNG */  "var index out of range (1 to # of var)",
        /* ERR_DUP_VAR      */  "duplicate variable id",
        /* ERR_LOWER_RNG    */  "lower bound out of range (0 to radix-1)",
        /* ERR_UPPER_RNG    */  "upper bound out of range (0 to radix-1)",
        /* ERR_UPPER_LOWER  */  "upper bound must be >= lower bound",
        /* ERR_UNEXP_EOF    */  "unexpected end of file",
                                NULL
    };

    code = abs(code);
    printf("%s\n%s\n",errors[code],expression);
    exit(1);
}

next_token()
```

```
{
    token_type = yylex();
    if (token_type == TOK_SYNERR) {
        syntax_error(TOK_SYNERR);
    }
    return(token_type);
}

match(s)
char    *s;
{
    if (!strcmp(s,yytext))
        return(YES);
    else
        return(NO);
}
```

```
/* dm.c

    - This module implements the Dueck & Miller heuristic for minimizing
    an expression.

*/

#include "defs.h"

Dueck_Miller()
/* ---------------------------------------------------------------------
    :function:
        - Perform the Dueck & Miller heuristic on the input expression
    :algorithm:
        Start with a working copy E_work of the original function E_orig;
        Initialize a final function E_final;
        While (there are still minterms to pick) {
            Pick a minterm X from E_work;
            Pick the best implicant I for X;
            Subtract I from E_work;
            Add I to E_final;
        }
    :globals:
        E_orig
        e_flag
        m_flag
        q_flag
        G_flag
        FO_ratio
    :side_effects:
        STAT
        HEUR
        E_work
        E_final[]
    :called_by:
        main()
    :calls:
        dealloc_expr()
        dup_expr()
        print_terms()
        print_map()
        mim()
        pick_implicant()
        subtract_implicant()
        print_source()
--------------------------------------------------------------------------- */
{
    int num_impl = 0;
    int *X;
    Term    *I;
```

```
    float   ratio;

    if (E_final[D_M].T != NULL)
        dealloc_expr(&E_final[D_M]);

    STAT = &DM_stat;
    HEUR = D_M;
    dup_expr(&E_work,&E_orig);
    E_final[HEUR].nterm = 0;
    E_final[HEUR].radix = E_orig.radix;
    E_final[HEUR].nvar = E_orig.nvar;
    E_final[HEUR].T = NULL;

    if (e_flag)
        print_terms(&E_orig);
    if (m_flag) {
        printf(" Orig map (D&M):\n");
        print_map();
    }
    for (;;) {
        if ((X = mim(&E_work)) == NULL)
            break;
        I = pick_implicant(X);
        num_impl++;
        subtract_implicant(I);
        if (i_flag)
            print_implicant("D&M",X,I);
        if (m_flag)
            print_map();
    }
    ratio = ((float)num_impl/(float)E_orig.nterm);
    if (FO_ratio != 0.0) {
        if (ratio > FO_ratio) {
            print_source(&E_orig,"D&M",num_impl);
        }
    }
    if (!q_flag && !G_flag) {
        printf("  D&M: %4d/%-4d implicants - %4.2f%%\n\n",
            num_impl,E_orig.nterm,ratio);
    }
    dealloc_expr(&E_work);
}
```

```
static int cf(E,x)
Expression *E;
int x;
/* ---------------------------------------------------------------------------
    :function:
        - Determine the clustering factor for (x1,x2,x3,x4....xn)
        - Builds a vector of coordinates for _cf()
        - Local to dm.c
    :calls:
        _cf()
    :returns:
        - An integer clustering factor
--------------------------------------------------------------------------- */
{
    int *X;

    X = (int *) &x;
    return(_cf(E,X));
}


static int _cf(E,X)
Expression  *E;
int *X;
/* ---------------------------------------------------------------------------
    :function:
        - Compute the clustering factor at X, where X is a vector of
        coordinates.
        - Local to dm.c
    :globals:
        radix
        nvar
    :side_effects:
        STAT
    :called_by:
        mim()
    :calls:
        _eval()
        vcopy()
    :returns:
        - An integer clustering factor
--------------------------------------------------------------------------- */
{
    int nterm = E->nterm;
    register i,j,k;
    int value[2],
        expanded,
        val1[2],
        val2[2];
    int cf = 0,
        dea = 0,
        ea = 0;

    STAT->calls_cf++;
    vcopy(value,_eval(E,X));
    if ((value[EVAL] < 1) || (value[EVAL] >= radix))
        return(MAX_INT);

    /* for each variable (direction)... */
    for (i=0; i < nvar; i++ ) {
        expanded = 0;
        /* If not on a left hand edge, move left */
        if (X[i]) {
            X[i]--;
            vcopy(val1,_eval(E,X));
```

```
            if (val1[EVAL] >= value[EVAL]) {
                expanded++;
                ea++;
            }
            X[i]++;
        }

        /* if we didn't start on a right hand edge, move right */
        if (X[i] < (radix-1)) {
            X[i]++;
            vcopy(val2, _eval(E,X));
            if (val2[EVAL] >= value[EVAL]) {
                expanded++;
                ea++;
            }
            X[i]--;
        }
        if (expanded)
            dea++;
    }
    /* compute the clustering factor */

    cf = (dea * (radix-1)) + ea;
    return(cf);
}
```

```
static int  *mim(E)
Expression *E;
/* ---------------------------------------------------------------------------
    :function:
        - Find the Most Isolated Minterm in the expression pointed to
        by E, and return its coordinates as a vector.
        - Local to dm.c
    :globals:
        nvar
    :side_effects:
        STAT
    :called_by:
        Dueck_Miller()
    :calls:
        next_coord()
        _eval()
        vcopy()
        _cf()
    :returns:
        - A vector of integers representing the coordinate of the most
        isolated minterm, or NULL if no more minterms.
        - The value at that location is also returned as the last integer
        in the vector.
--------------------------------------------------------------------------- */
{
    int cur_val = E->radix,
        cur_CF = MAX_INT,
        cf,
        value[2],
        term;
    int *X,*next_coord(),i;
    static int save_coord[MAX_VAR+1];

    STAT->calls_mim++;
    for (term=0; term < E->nterm; term++) {
        i = 1;
        while ((X=next_coord(&(E->T[term])),i)) != NULL) {
            vcopy(value, _eval(E,X));
            if (value[EVAL] > 0) {
                if (value[EVAL] < cur_val) {
                    cur_val = value[EVAL];
                    for (i=0; i < E->nvar; i++) save_coord[i] = X[i];
                    cur_CF = _cf(E,X);
                }
                else if (value[EVAL] == cur_val) {
                    cf = _cf(E,X);
                    if (cf < cur_CF) {
                        cur_CF = cf;
                        for (i=0; i < E->nvar; i++) save_coord[i] = X[i];
                    }
                }
            }
            i = 0;
        }
    }
    if (cur_CF == MAX_INT)
        return(NULL);
    save_coord[E->nvar] = cur_val;
    return(save_coord);
}
```

```
static int  valid_implicant(I)
Term    *I;
/* ---------------------------------------------------------------------------
    :function:
        - Decide upon the validity of implicant I
        - Local to dm.c
    :globals:
        E_work
        E_orig
    :side_effects:
        STAT
    :called_by:
        pick_implicant()
    :calls:
        next_coord()
        _eval()
        vcopy()
    :returns:
        1 if a valid implicant
        0 if not
--------------------------------------------------------------------------- */
{
    int *X;
    int init = 1;
    int value = I->coeff;
    int Vo[2],Vw[2];

    STAT->calls_valid_implicant++;
    while ((X = next_coord(I,init)) != NULL) {
        init = 0;
        vcopy(Vw,_eval(&E_work,X));
        vcopy(Vo,_eval(&E_orig,X));
        if (((Vw[EVAL] < value) && !Vw[HLV]) && (Vo[EVAL] < (radix-1)))
            return(0);
    }
    return(1);
}
```

```
static int  compute_rbc(I)
Term    *I;
/* ---------------------------------------------------------------------------
    :function:
        - Compute the RBC for the given implicant
        - Local to dm.c
    :globals:
        radix
        nvar
    :side_effects:
        STAT
    :called_by:
        pick_implicant()
    :calls:
        next_coord()
        _eval()
        vcopy()
    :returns:
        - an integer RBC
--------------------------------------------------------------------------- */
{
    int *X;
    int I_value = I->coeff;
    register    i;
    int value[2],
        neighbor_value[2],
        good,
        bad,
        rbc = 0,
        init = 1;

    STAT->calls_compute_rbc++;
    /* for each coordinate in the implicant ... */
    while ((X = next_coord(I,init)) != NULL) {
        init = 0;
        vcopy(value,_eval(&E_work,X));
        if (value[EVAL] == radix)
            continue;
        /* for each direction ... */
        for (i=0; i < nvar; i++) {
            good = 0;
            bad = 0;
            if (value[EVAL] == I_value)
                good = 1;

            /* if there is a left neighbor, examine it */
            if ((X[i] != 0) && (X[i] == I->B[i].lower)) {
                X[i]--;
                vcopy(neighbor_value,_eval(&E_work,X));
                X[i]++;
                if (neighbor_value[EVAL] != 0) {
                    if (neighbor_value[EVAL] == (value[EVAL] - I_value))
                        good = 1;
                    if (neighbor_value[EVAL] == value[EVAL])
                        bad = 1;
                }
            }

            /* if there is a right neighbor, examine it */
            if ((X[i] != (radix-1)) && (X[i] == I->B[i].upper)) {
                X[i]++;
                vcopy(neighbor_value,_eval(&E_work,X));
                X[i]--;
                if (neighbor_value[EVAL] != 0) {
                    if (neighbor_value[EVAL] == (value[EVAL] - I_value))
```

```
                        good = 1;
                    if (neighbor_value[EVAL] == value[EVAL])
                        bad = 1;
                }
            }
            /* update the rbc */
            rbc = (rbc - good) + bad;
        }
    }
    return(rbc);
}
```

```
static Term *pick_implicant(X)
int     *X;
/* ---------------------------------------------------------------------------
    :function:
        - Pick the best implicant for minterm X
    :globals:
        radix
    :side_effects:
        STAT
    :called_by:
        Dueck_Miller()
    :calls:
        init_implicant()
        gen_bounds()
        next_implicant()
        _eval()
        vcopy()
        compue_rbc()
        copy_implicant()
        valid_implicant()
    :returns:
        - A pointer to a term representing the best implicant.
--------------------------------------------------------------------------- */
{
    int cur_rbc = MAX_INT,
        rbc = 0;
    Implicant   *I;
    static Bound    I_bound[MAX_VAR+1];
    static Term I_best;
    Bound   *B;
    int V[2];

    STAT->calls_pick_implicant++;
    I_best.B = I_bound;
    init_implicant(X);
    B = gen_bounds(X);
    while ((I = next_implicant(B)) != NULL) {
        vcopy(V, _eval(&E_orig,X));
        if (V[EVAL] == (radix-1)) {
            for (I->coeff = X[nvar]; I->coeff < radix; (I->coeff)++) {
                if (valid_implicant(I)) {
                    rbc = compute_rbc(I);
                    if (rbc < cur_rbc) {
                        cur_rbc = rbc;
                        copy_implicant(&I_best,I);
                    }
                }
            }
        }
        else {
            I->coeff = X[nvar];
            if (valid_implicant(I)) {
                rbc = compute_rbc(I);
                if (rbc < cur_rbc) {
                    cur_rbc = rbc;
                    copy_implicant(&I_best,I);
                }
            }
        }
    }
    return(&I_best);
}
```

```
/* pa.c

    - This module implements the Pomper-Armstrong heuristic for minimizing
    an expression.

*/

#include "defs.h"

Pomper_Armstrong()
/* ---------------------------------------------------------------------------
    :function:
        - Perform the Pomper & Armstrong heuristic on the input expression
    :globals:
        FO_ratio
        E_orig
        e_flag
        m_flag
        q_flag
        i_flag
        G_flag
    :side_effects:
        STAT
        HEUR
        E_work
        E_final[]
    :called_by:
        main()
    :calls:
        dealloc_expr()
        dup_expr()
        print_terms()
        print_map()
        mim()
        pick_implicant()
        subtract_implicant()
        print_source()
--------------------------------------------------------------------------- */
{
    int num_impl = 0;
    int *X;
    Term    *I;
    float   ratio;

    if (E_final[P_A].T != NULL)
        dealloc_expr(&E_final[P_A]);

    STAT = &PA_stat;
    HEUR = P_A;
    dup_expr(&E_work,&E_orig);
```

```
    E_final[HEUR].nterm = 0;
    E_final[HEUR].radix = E_orig.radix;
    E_final[HEUR].nvar = E_orig.nvar;
    E_final[HEUR].T = NULL;

    if (e_flag)
        print_terms(&E_orig);
    if (m_flag) {
        printf(" Orig map (F&A):\n");
        print_map();
    }
    for (;;) {
        if ((X = mim(&E_work)) == NULL)
            break;
        I = pick_implicant(X);

        num_impl++;
        subtract_implicant(I);
        if (i_flag)
            print_implicant("F&A",X,I);
        if (m_flag)
            print_map();
    }
    ratio = ((float)num_impl/(float)E_orig.nterm);
    if (FO_ratio != 0.0) {
        if (ratio > FO_ratio) {
            print_source(&E_orig,"F&A",num_impl);
        }
    }
    if (!q_flag && !G_flag) {
        printf("  F&A: %4d/%-4d implicants - %4.2f%%\n\n",
            num_impl,E_orig.nterm,ratio);
    }
    dealloc_expr(&E_work);
}
```

```
static int  *mim(E)
Expression  *E;
/* ---------------------------------------------------------------------------
    :function:
        - Return a vector corresponding to the coordinates of the current
        Most Isolated Minterm
    :globals:
        radix
        nvar
    :side_effects:
        STAT
    :called_by:
        Pomper_Armstrong()
    :calls:
        next_coord()
        _eval()
        vcopy()
        random()
    :returns:
        - A vector of integers representing the current Most Isolated
        Minterm, or NULL if no more minterms.
        - The last integer in the vector is the value of the function at
        that coordinate.
--------------------------------------------------------------------------- */
{
    int weight = MAX_INT;
    int tmp;
    int *X,i,*next_coord();
    int first;
    static int  save_coord[MAX_VAR+1] = { -1 };
    int value[2];
    int term;

    STAT->calls_mim++;
    for (term = 0; term < E->nterm; term++) {
        first = 1;
        while ((X=next_coord(&(E->T[term]),first)) != NULL) {
            vcopy(value,_eval(E,X));
            if ((value[EVAL] > 0) && (value[EVAL] != radix)) {
                tmp = random();
                if (tmp < weight) {
                    for (i=0; i < nvar; i++)
                        save_coord[i] = X[i];
                    save_coord[nvar] = value[EVAL];
                    weight = tmp;
                }
            }
            first = 0;
        }
    }
    if (weight != MAX_INT)
        return(save_coord);
    return(NULL);
}
```

```
static int  random()
/* ----------------------------------------------------------------------------
    :function:
        - Return a number between 0 and 32766 using a linear congruential
        method
    :called_by:
        mim()
---------------------------------------------------------------------------- */
{
    static long a=100001;



     a = (a*125) % 2796203;
     return((int)(((float)a/2796203)*32767));
 )
```

```
static int  valid_implicant(I)
Term    *I;
/* ---------------------------------------------------------------------------
    :function:
        - Decide upon the validity of implicant I
    :globals:
        radix
    :side_effects:
        STAT
    :called_by:
        pick_implicant()
    :calls:
        next_coord()
        _eval()
        vcopy()
    :returns:
        num_zeroed if valid
        0 if not
--------------------------------------------------------------------------- */
{
    int *X;
    int init = 1;
    int num_zeroed = 0;
    int value = I->coeff;
    int Vo[2],Vw[2];

    STAT->calls_valid_implicant++;
    while ((X = next_coord(I,init)) != NULL) {
        init = 0;
        vcopy(Vw,_eval(&E_work,X));
        vcopy(Vo,_eval(&E_orig,X));
        if (((Vw[EVAL] < value) && !Vw[HLV]) && (Vo[EVAL] < (radix-1)))
            return(0);
        if (Vw[EVAL] <= value)
            num_zeroed++;
    }
    return(num_zeroed);
}
```

```
static Term *pick_implicant(X)
int     *X;
/* ---------------------------------------------------------------------------
    :function:
        - Pick the best implicant for minterm X
    :globals:
        nvar
        radix
    :side_effects:
        STAT
    :called_by:
        Pomper_Armstrong()
    :calls:
        init_implicant()
        gen_bounds()
        next_implicant()
        _eval()
        vcopy()
        copy_implicant()
    :returns:
        - A pointer to a Term representing the best implicant.
--------------------------------------------------------------------------- */
{
    int cur_num_zeroed = -1,
        cur_num_covered = -1;
    int num_covered,
        num_zeroed;
    Implicant   *I;
    static Bound    I_bound[MAX_VAR+1];
    static Term I_best;
    Bound   *B;
    int V[2];
    register i;

    STAT->calls_pick_implicant++;
    I_best.B = I_bound;
    init_implicant(X);
    B = gen_bounds(X);
    while ((I = next_implicant(B)) != NULL) {
        vcopy(V,_eval(&E_orig,X));
        if (V[EVAL] == (radix-1)) {
            for (I->coeff = X[nvar]; I->coeff < radix; I->coeff++) {
                if (num_zeroed = valid_implicant(I)) {
                    num_covered = 1;
                    for (i=0; i < nvar; i++)
                        num_covered *= (I->B[i].upper - I->B[i].lower + 1);
                    if ((num_zeroed > cur_num_zeroed) || (
                            (num_zeroed == cur_num_zeroed) &&
                            (num_covered = cur_num_covered)
                    )) {
                        cur_num_zeroed = num_zeroed;
                        cur_num_covered = num_covered;
                        copy_implicant(&I_best,I);
                    }
                }
            }
        }
        else {
            I->coeff = X[nvar];
            if (num_zeroed = valid_implicant(I)) {
                num_covered = 1;
                for (i=0; i < nvar; i++)
                    num_covered *= (I->B[i].upper - I->B[i].lower + 1);
                if ((num_zeroed > cur_num_zeroed) || (
                        (num_zeroed == cur_num_zeroed) &&
```

```
                        (num_covered = cur_num_covered)
                    )) {
                        cur_num_zeroed = num_zeroed;
                        cur_num_covered = num_covered;
                        copy_implicant(&I_best,I);
                    }
                }
            }
        }
        return(&I_best);
    }
```

```
/* common.c

    - Heuristic support functions common to D&M and F&A.

*/

#include "defs.h"

int *eval(E,x)
Expression  *E;
int x;
/* ---------------------------------------------------------------------------
    :function:
        - Evaluate the expression at (x1,x2,x3,x4,x5....xn)
        - Builds a vector of coordinates for _eval()
    :calls:
        _eval()
    :returns:
        - A vector containing the integer result 0 <= eval() <= radix
        and a flag set if the value reached the highest logic value
--------------------------------------------------------------------------- */
{
    int *X;

    X = (int *) &x;
    return(_eval(E,X));
}

int *_eval(E,X)
Expression  *E;
int *X;
/* ---------------------------------------------------------------------------
    :function:
        - Evaluate the expression at X, where X is a vector of coordinates
    :globals:
        nvar
        radix
    :side_effects:
        STAT
    :called_by:
        mim() - pa.c
        valid_implicant() - pa.c
        pick_implicant() - pa.c
        mim() - dm.c
        valid_implicant() - dm.c
        pick_implicant() - dm.c
        _cf()
        compute_rbc()
        eval()
        gen_bounds()
        print_map()
    :returns:
```

```
        - A vector with the value of the expression at the specified
        coordinate as its first element, and a flag set if this value
        has attained the highest logic value (HLV)
-------------------------------------------------------------------------- */
{
    int nterm = E->nterm;
    register i,j,k;
    int out_of_bounds;
    static int  V[2];
    register    rm1 = radix-1;

    STAT->calls_eval++;
    V[EVAL] = 0;
    V[HLV] = 0;
    /* for each term ... */
    for (i=0; i < nterm; i++) {
        /* for each variable ... */
        for (j=0,out_of_bounds=0; j < nvar; j++) {
            if (
                (X[j] < E->T[i].B[j].lower) ||
                (X[j] > E->T[i].B[j].upper)
            ) {
                out_of_bounds = 1;
                break;
            }
        }
        if (out_of_bounds)
            continue;

        /* if this is a don't care, return the radix */
        if (E->T[i].coeff == radix) {
            V[EVAL] = radix;
            return(V);
        }

        V[EVAL] += E->T[i].coeff;
        if (V[EVAL] >= rm1) {
            /* set a flag which means E_orig was saturated at this X */
            V[HLV] = 1;
        }
        if (V[EVAL] > rm1) {
            V[EVAL] = rm1;
        }
        else if (V[HLV] && (V[EVAL] <= 0)) {
            V[EVAL] = radix;
            return(V);
        }
    }
    return(V);
}
```

```
int *next_coord(T,first)
Term    *T;
int first;
/* ---------------------------------------------------------------------------
    :function:
        - Compute the next possible coordinate for term *T
        - If first == 1, initialize the coord vector
    :called_by:
        mim() - pa.c
        valid_implicant() - pa.c
        mim() - dm.c
        valid_implicant() - dm.c
        compute_rbc()
    :returns:
        - An integer vector containing the coordinates.
--------------------------------------------------------------------------- */
{
    static int  coord[MAX_VAR+1];
    static i;

    /* if the first time through, load the vector */
    if (first) {
        for (i=0; i < nvar; i++) {
            coord[i] = T->B[i].lower;
        }
    }
    else {
        i = 0;
        coord[i]++;
        for (;;) {
            if (coord[i] > T->B[i].upper) {
                coord[i] = T->B[i].lower;
                i++;
                if (i >= nvar)
                    return(NULL);
                coord[i]++;
            }
            else {
                break;
            }
        }
    }
    return(coord);
}
```

```
Bound   *gen_bounds(X)
int *X;
/* ---------------------------------------------------------------------------
    :function:
        - Generate the permissible bounds around location X in the
        working expression
    :globals:
        radix
        nvar
        E_work
        E_orig
    :side_effects:
        STAT
    :called_by:
        pick_implicant() - pa.c
        pick_implicant() - dm.c
    :calls:
        _eval()
        vcopy()
    :returns:
        - A bounds array
--------------------------------------------------------------------------- */
{
    static Bound    B[MAX_VAR+1];
    int nterm = E_work.nterm;
    register i,j,k;
    int value,Vw[2],Vo[2];
    int Xp[MAX_VAR+1];

    value = X[nvar];

    STAT->calls_gen_bounds++;
    /* for each variable (direction)... */
    for (i=0; i < nvar; i++ ) {
        /* dup the coordinate */
        for (j=0; j < nvar; j++) Xp[j] = X[j];
        B[i].lower = X[i];
        /* while not on a left hand edge, move left */
        while (Xp[i] > 0) {
            Xp[i]--;
            vcopy(Vw,_eval(&E_work,Xp));
            vcopy(Vo,_eval(&E_orig,Xp));
            /* if can't expand to left .... */
            if (!((value > Vw[EVAL]) && (Vo[EVAL] < (radix-1)))) {
                B[i].lower = Xp[i];
            }
            else
                break;
        }
```

```
        /* dup the coordinate */
        for (j=0; j < nvar; j++) Xp[j] = X[j];
        B[i].upper = X[i];
        /* while not on a right hand edge, move right */
        while (Xp[i] < (radix-1)) {
            Xp[i]++;
            vcopy(Vw,_eval(&E_work,Xp));
            vcopy(Vo,_eval(&E_orig,Xp));
            /* if can't expand to right ... */
            if (!((value > Vw[EVAL]) && (Vo[EVAL] < (radix-1)))) {
                B[i].upper = Xp[i];
            }
            else
                break;
        }


    }
    return (B);
}
```

```
    /* Working structures for picking the next implicant within bounds */

static Bound    IB[MAX_VAR+1];  /* Current bounds */
static Term I;                  /* Implicant */
static int
    I_var,
    I_first,
    I_val;
int X_orig[MAX_VAR+1];          /* Where we start */

init_implicant(X)
int     *X;
/* ---------------------------------------------------------------------------
    :function:
        - Initialize the static term structure above from which successive
        implicants will be returned
        - X is the starting minterm
    :side_effects:
        - The structures above
    :called_by:
        pick_implicant() - pa.c
        pick_implicant() - dm.c
--------------------------------------------------------------------------- */
{
    int nterm = E_work.nterm;
    register i;

    /* initialize the implicant */
    I.B = IB;
    I.coeff = X[nvar];
    for (i=0; i < nvar; i++) {
        I.B[i].upper = X[i];
        I.B[i].lower = X[i];
    }
    I_var = 0;
    I_first = 1;
    I_val = X[nvar];
    for (i=0; i < nvar; i++) X_orig[i] = X[i];
}


Term    *next_implicant(B)
Bound   *B;
/* ---------------------------------------------------------------------------
    :function:
        - On each call, return the next implicant within bounds B
    :side_effects:
        STAT
    :called_by:
        pick_implicant() - pa.c
        pick_implicant() - dm.c
    :returns:
        - An implicant as a term structure
--------------------------------------------------------------------------- */
{
    int nterm = E_work.nterm;
    int Xp[MAX_VAR+1];

    STAT->calls_next_implicant++;
    if (I_first) {
        I_first = 0;
        return(&I);
    }

    while (I_var < nvar) {
```

```
        /* expand left */
        I.B[I_var].lower--;

        /* if we can't go further left, then ... */
        if (I.B[I_var].lower < B[I_var].lower) {

            /* move back and go right */
            I.B[I_var].lower = X_orig[I_var];
            I.B[I_var].upper++;

            /* if we can't go further right, then ... */
            if (I.B[I_var].upper > B[I_var].upper) {

                /* reset and go to the next higher dimension */
                I.B[I_var].upper = X_orig[I_var];
                I_var++;
                continue;
            }
        }
        I_var = 0;

        return(&I);
    }
    return(NULL);
}
```

```
int copy_implicant(dest,src)
Term    *dest,*src;
/* ---------------------------------------------------------------------------
    :function:
        - Copy the implicant pointed to by src to dest
    :called_by:
        pick_implicant() - pa.c
        pick_implicant() - dm.c
--------------------------------------------------------------------------- */
{
    register i;

    dest->coeff = src->coeff;
    for (i=0; i < nvar; i++) {
        dest->B[i].lower = src->B[i].lower;
        dest->B[i].upper = src->B[i].upper;
    }
}


subtract_implicant(I)
Term    *I;
/* ---------------------------------------------------------------------------
    :function:
        - Add implicant I to the working expression as a negative term
        (negated coefficient)
        - Add implicant I to tthe final expression
    :globals:
        HEUR
        nvar
    :side_effects:
        E_work
        E_final[]
--------------------------------------------------------------------------- */
{
    register i,term;

    term = E_work.nterm;
    E_work.nterm++;
    E_work.T = alloc_term(E_work.T,-(I->coeff),E_work.nterm);
    for (i=0; i < nvar; i++) {
        E_work.T[term].B[i].lower = I->B[i].lower;
        E_work.T[term].B[i].upper = I->B[i].upper;
    }

    term = E_final[HEUR].nterm;
    E_final[HEUR].nterm++;
    E_final[HEUR].T = alloc_term(E_final[HEUR].T,I->coeff,E_final[HEUR].nterm);
    for (i=0; i < nvar; i++) {
        E_final[HEUR].T[term].B[i].lower = I->B[i].lower;
        E_final[HEUR].T[term].B[i].upper = I->B[i].upper;
    }
}

/* vcopy()

    - copies the value vector from s to d
*/
vcopy(d,s)
int *d,*s;
{
    d[0] = s[0];
    d[1] = s[1];
}
```

```
#include "defs.h"

/* memory allocation functions ---------------------------------------- */

Term *alloc_term(p,coeff,n)
Term *p;
int coeff,n;
/* ------------------------------------------------------------------------
    :function:
        - Allocate space for a term array, initializing the last element
        - If p is NULL, allocate new space
        - If p is not, realloc
    :returns:
        - A pointer to the Term
------------------------------------------------------------------------ */
{
    char    *malloc(),*realloc();
    Bound *alloc_bound();

    if (p == NULL) {
        if ((p=(Term *)malloc(sizeof(Term)*n)) == NULL)
            fatal("alloc_term(): out of memory\n");
        p->coeff = coeff;
        p->B = alloc_bound();
    }
    else {
        if ((p=(Term *)realloc(p,sizeof(Term)*n)) == NULL)
            fatal("alloc_Term(): out of memory\n");
        p[n-1].coeff = coeff;
        p[n-1].B = alloc_bound();
    }
    return(p);
}

Bound *alloc_bound()
/* ------------------------------------------------------------------------
    :function:
        - Allocate space for E_orig.nvar bounds entries and initialize
        each bound to -1,E_orig.radix-1.
        - If p is NULL, allocate new space
    :globals:
        E_orig
    :returns:
        - A pointer to the Bound array
------------------------------------------------------------------------ */
{
    Bound *p;
    char    *malloc();
    register i;
```

```
    if ((p=(Bound *)malloc(sizeof(Bound)*(E_orig.nvar))) == NULL)
        fatal("alloc_bound(): out of memory\n");

    for (i=0; i < E_orig.nvar; i++) {
        p[i].lower = -1;
        p[i].upper = E_orig.radix-1;
    }

    return(p);
}
```

```
init_expr()
/* ---------------------------------------------------------------------------
    :function:
        - Initialize E_work, E_orig and E_final
    :side_effects:
        E_work
        E_orig
        E_final
--------------------------------------------------------------------------- */
{
    E_work.T = NULL;
    E_orig.T = NULL;
    E_orig.nvar = 0;
    E_orig.nterm = 0;
    E_orig.radix = 0;
    E_final[0].T = NULL;
    E_final[1].T = NULL;
}


dealloc_expr(e)
Expression  *e;
/* ---------------------------------------------------------------------------
    :function:
        - Deallocate the expression pointed to by e
--------------------------------------------------------------------------- */
{
    Term    *p;
    register    i;

    if (e->T != NULL) {
        for (p = e->T,i=0; i < e->nterm; i++)
            if (p[i].B != NULL)
                free(p[i].B);
        free(p);
    }
}
```

```
dup_expr(E_dest,E_src)
Expression  *E_dest,*E_src;
/* ---------------------------------------------------------------------------
    :function:
        - Duplicate the expression pointed to by E_src by allocating
        and copying into the expression pointed to by E_dest.
    :calls:
        alloc_bound()
--------------------------------------------------------------------------- */
{
    Term    *T;
    Bound   *B;
    register    i,j;
    char    *malloc();

    E_dest->radix = E_src->radix;
    E_dest->nvar = E_src->nvar;
    E_dest->nterm = E_src->nterm;

    if ((T=(Term *)malloc(sizeof(Term)*(E_dest->nterm))) == NULL)
        fatal("dup_expr(): out of memory\n");

    for (i=0; i < E_src->nterm; i++) {
        T[i].coeff = E_src->T[i].coeff;
        T[i].B = alloc_bound();
        for (j=0; j < E_src->nvar; j++) {
            T[i].B[j].lower = E_src->T[i].B[j].lower;
            T[i].B[j].upper = E_src->T[i].B[j].upper;
        }
    }
    E_dest->T = T;
}
```

```
/* mvltest.c

    - A test source file generator
*/

#include <stdio.h>

char
*version = "\nMVLTEST  V1.0  Copyright 1988 Naval Postgraduate School\n\n",
*usage = "usage: mvltest [-sN -rN -iN -tN -oN]\n",
*help[] = {
    "    -sN   - Cycle the random seed N times (default = 0)",
    "    -rN   - Set radix to N (default = 4)",
    "    -iN   - Set number of inputs to N (default = 2)",
    "    -tN   - Set number of terms to N (default = 1)",
    "    -oN   - Set number of output to N (default = 1)",
    NULL
};

main(argc,argv)
char    *argv[];
{
    int nterms = 1;
    int reps = 1,
        radix = 4,
        nvar = 2,
        coeff = 1;
    int seed = 0;
    int i,j,k;
    int *b,*get_bound();
    float random();
    char    *p;
    register ap;


    for (ap=1; ap < argc; ap++) {
        p = argv[ap];
        if (*p++ = '-') {
            while (*p) {
                switch (*p++) {
                case 'r':
                    sscanf(p,"%d",&radix);
                    *p = '\0';
                    break;
                case 'i':
                    sscanf(p,"%d",&nvar);
                    *p = '\0';
                    break;
                case 't':
                    sscanf(p,"%d",&nterms);
```

```
                    *p = '\0';
                    break;
                case 'o':
                    sscanf(p,"%d",&reps);
                    *p = '\0';
                    break;
                case 's':
                    sscanf(p,"%d",&seed);
                    *p = '\0';
                    break;
                case '-':
                    fprintf(stderr,"%s%s",version,usage);
                    for (ap=0; help[ap]; ap++)
                        fprintf(stderr,"%s\n",help[ap]);
                    exit(0);
                default:
                    fprintf(stderr,"mvltest: unknown switch - %c\n",p[-1]);
                    fprintf(stderr,"%s",usage);
                    exit(1);
                }
            }
        }
    }
    for (i=0; i < seed; i++) random();
    for (i = 0; i < reps; i++) {
        printf("%d:%d:",radix,nvar);
        for (j=0; j < nterms; j++) {
            printf("\n   +%d",(int)(random()*(float)(radix-1))+1);
            for (k=1; k <= nvar; k++) {
                b = get_bound(radix);
                printf("*X%d(%d,%d)",k,b[0],b[1]);
            }
        }
        printf(";\n");
    }
}


int *get_bound(radix)
{
    static int b[3];
    float   random();

    for (;;) {
        b[0] = (int)(random()*((float)(radix)-0.001));
        b[1] = (int)(random()*((float)(radix)-0.001));
        if (b[0] <= b[1])
            break;
    }
    return(b);
}

float random()
{
    static long a=100001;

    a = (a*125) % 2796203;
    return((float)a/2796203);
}
```

```
# Makefile for mvl
#
CFLAGS=-O
OBJS=main.o dm.o pa.o common.o parse.o alloc.o

all: mvl mvltest

mvl: $(OBJS)
	cc $(CFLAGS) $(OBJS) -o mvl -ll

mvltest: mvltest.o
	cc $(CFLAGS) mvltest.o -o mvltest

main.o: main.c
alloc.o: alloc.c
parse.o: parse.c
dm.o: dm.c
pa.o: pa.c
common.o: common.c
parse.c: parse.l
	lex parse.l
	mv -f lex.yy.c parse.c

mvltest.o: mvltest.c
```

LIST OF REFERENCES

1. Kerkhoff, H. G. and Butler, J. T., "A Module Compiler for High-Radix CCD-PLA's", Preprint, submitted to VLSI Integration.
2. Kawahito, S., Kameyama, M., Higuchi, T. and Yamada, H., *A Multiplier Chip With Multiple-Valued Bi-Directional Current-Mode Logic Circuits*, Proceedings of the 1987 International Symposium on Multiple-Valued Logic, pp. 172-180, May 1987.
3. Gamal, A. E., et. al., *A CMOS 32b Wallace Tree Multiplier-Accumulator*, Dig. IEEE Int. Solid-State Circuit Conference, THPM 15.5, 1986.
4. Brayton, R. K., Hachtel, G. D., McMullen, C. T., and Sangiovanni-Vincentelli, A., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic, Boston, Massachusetts, 1984.
5. Pomper, G. and Armstrong, J. A., *Representation of Multivalued Functions Using the Direct Cover Method*, IEEE Transactions on Computing, September 1981, pp. 674-679.
6. Dueck, G. W. and Miller, D. M., *A Direct Cover MVL Minimization Using the Truncated Sum*, Proceedings of the 17th International Symposium on Multiple-Valued Logic, May 1987, pp. 221-227.
7. Tirumalai, P., and Butler, J. T., *Analysis of Minimization Algorithms For Multiple-Valued Programmable Logic Arrays*, Proceedings of the 18th International Symposium on Multiple-Valued Logic, May 1987, pp. 226-236.
8. Ko, Y. H., *Design of Multiple-Valued Programmable Logic Array*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.

BIBLIOGRAPHY

Butler, J. T. and Kerkhoff, H. S., *Multiple-Valued CCD Circuits*, Computer, Volume 21, #24, April 1988, pp. 58-69.

Current, K. W., *High Density Integrated Computing Circuitry With Multiple-Valued Logic*, IEEE Journal of Solid State Circuits, Vol. SC-15, February 1980.

Onneweer, S. P. and Kerkhoff, H. G., *High-Radix Current-Mode CMOS Circuits Based on the Truncated-Difference Operator*, Proceedings of the International Symposium on Multiple-Valued Logic, May 1987, pp. 188-195.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	John P. Powers, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Jon T. Butler, Code 62Bu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	5
5.	Chyan Yang, Code 62Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	LCDR John M. Yurchak, Code 52Yu Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	George J. Thaler, Code 62Tr Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1

8. Dr. George Abraham, Code 1005 — 1
Office of Research and Technology, Applied Physics
Naval Research Laboratories
4555 Overlook Avenue S.W.
Washington, D.C. 20375

9. Dr. Robert Williams — 1
Naval Air Development Center, Code 505B
Warminster, Pennsylvania 18974-5000

10. Dr. Ir. Hans G. Kerkhoff — 1
IC Technology and Electronics Group
Department of Electrical Engineering
University of Twente
7500AE Enschede, The Netherlands

11. Dr. Joo-kang Lee — 1
POSTECH Research Institute of Science and Technology
P.O. Box 125
Pohang City, Kyungbak 680
Republic of Korea

12. Dr. Ir. Siep Onneweer — 1
IC Technology and Electronics Group
Department of Electrical Engineering
University of Twente
7500AE Enschede, The Netherlands

13. CAPT W. P. Averill — 1
U. S. Naval Academy
Department of Electrical Engineering
Annapolis, Maryland 21402

14. Naval Academy Library — 2
Chinhae, Kyungnam 602-00
Republic of Korea

15. CDR Cho, Duck-Woon — 1
Naval Academy
Chinhae, Kyungnam 602-00
Republic of Korea

16. CDR Baek, Chil-Ki 1
Naval Academy
Chinhae, Kyungnam 602-00
Republic of Korea

17. LCDR Kwon, Yong-Soo 1
Naval Academy
Chinhae, Kyungnam 602-00
Republic of Korea

18. CAPT Yim, Man-Taek 1
P.O. Box 201-01
Computer Center Naval Headquarters
Sin Kil 7 Dong, Young Dong Po Gu
Seoul 150-09
Republic of Korea

19. CDR Kim, Kook-Bo 1
P.O. Box 201-01
Computer Center Naval Headquarters
Sin Kil 7 Dong, Young Dong Po Gu
Seoul 150-09
Republic of Korea

20. LT Han, Kyung-Sub 1
P.O. Box 201-01
Computer Center Naval Headquarters
Sin Kil 7 Dong, Young Dong Po Gu
Seoul 150-09,
Republic of Korea

21. LT Lee, Hoon-Seop 2
1069-42 Nam-Hyeon Dong
Kwan-AK Gu, Seoul 151-00
Republic of Korea

Thesis
L3855 Lee
c.1 A CAD tool for current-mode multiple-valued CMOS circuits.